MORRIS, NICHOLS, ARSHT & TUNNELL LLP

1201 NORTH MARKET STREET
P.O. BOX 1347
WILMINGTON, DELAWARE 19899-1347

302 658 9200
302 658 3989 FAX

RODGER D. SMITH II
302 351 9205
302 498 6209 FAX
rsmith@mnat.com

January 10, 2007

BY E-FILING

Dr. Peter T. Dalleo
Clerk of the Court
United States District Court
844 N. King Street
Wilmington, DE 19801

Re: *Intermec IP Corp. v. Alien Technology Corp.*
C.A. No. 06-411 (GMS)

Dear Dr. Dalleo:

Attached please find the following patents that were inadvertently omitted from the Amended Complaint (D.I. 23) that was filed on January 8, 2007: U.S. Patent No. 5,528,222 (Ex. A); U.S. Patent No. 5,828,318 (Ex. B); U.S. Patent No. 6,286,762 (Ex. C); and U.S. Patent No. 6,812,852 (Ex. D).

Respectfully,

/s/ Rodger D. Smith II

Rodger D. Smith II (#3778)

cc: Frederick L. Cottrell III, Esquire (By E-Filing)
Gregory P. Stone, Esquire (By Facsimile)

662575

EXHIBIT A

US005528222A

United States Patent [19]

Moskowitz et al.

[11] **Patent Number: 5,528,222**

[45] **Date of Patent: Jun. 18, 1996**

[54] **RADIO FREQUENCY CIRCUIT AND MEMORY IN THIN FLEXIBLE PACKAGE**

[75] Inventors: **Paul A. Moskowitz**, Yorktown Heights; **Michael J. Brady**, Brewster; **Paul W. Coteus**, Yorktown Heights, all of N.Y.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **303,977**

[22] Filed: **Sep. 9, 1994**

[51] **Int. Cl.**[6] **H04Q 1/02**

[52] **U.S. Cl.** **340/572**; 29/825; 29/829; 29/836; 340/825.3; 340/825.34; 340/825.54

[58] **Field of Search** 340/572, 825.34, 340/825.3, 825.54; 29/836, 829, 825

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,934,122	1/1976	Riccitelli	340/825.34
4,009,375	2/1977	White et al.	364/436
4,674,175	6/1987	Stampfli	437/209
4,818,973	4/1989	Yamakawa et al.	340/572
4,857,893	8/1989	Carroll	340/572
5,014,040	5/1991	Weaver et al.	340/572
5,204,663	4/1993	Lee	340/825.3
5,257,011	10/1993	Beigel	340/572
5,396,218	3/1995	Olah	340/572

FOREIGN PATENT DOCUMENTS

0481776	4/1992	European Pat. Off. .
0595549	5/1994	European Pat. Off. .
4319878	12/1993	Germany .
2173888	9/1990	Japan .
5266268	10/1993	Japan .
9309551	5/1993	WIPO .
9411846	5/1994	WIPO .

OTHER PUBLICATIONS

Patent Abstracts of Japan, vol. 13, No. 560 (M–906) 13 Dec. 1989 & JP, A, 01 234 296 (NEC Corp.) 19 Sep. 1989.

International Standard 7810, "Identification cards–Physical characteristics" First Edition–1985–12–15.

R. R. Tumalla et al, "Microelectronics Packaging Handbook", 1989, pp. 68, 76, 1154.

Primary Examiner—Glen Swann
Attorney, Agent, or Firm—Louis J. Percello

[57] **ABSTRACT**

A novel thin and flexible radio frequency (RF) tag has a semiconductor circuit with logic, memory, and a radio frequency circuits, connected to an antenna with all interconnections placed on a single plane of wiring without crossovers. The elements of the package (substrate, antenna, and laminated covers) are flexible. The elements of the package are all thin. The tag is thin and flexible, enabling a unique range of applications including: RF ID tagging of credit cards, passports, admission tickets, and postage stamps.

29 Claims, 10 Drawing Sheets

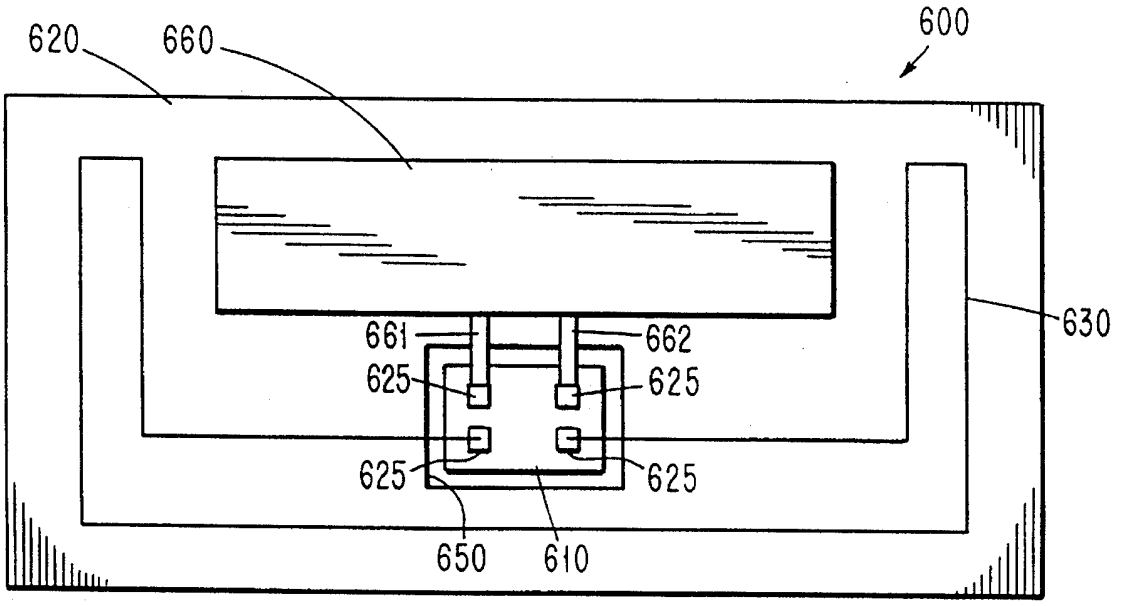


FIG. 1A
PRIOR ART

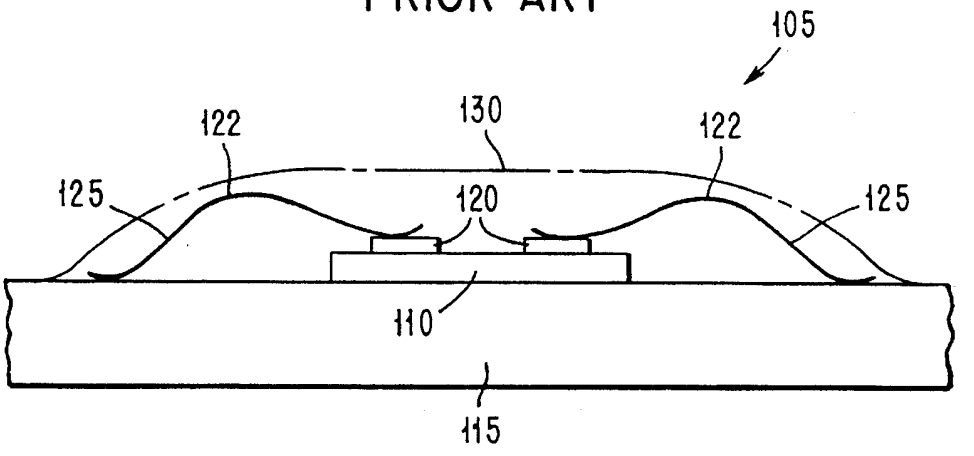


FIG. 1B
PRIOR ART

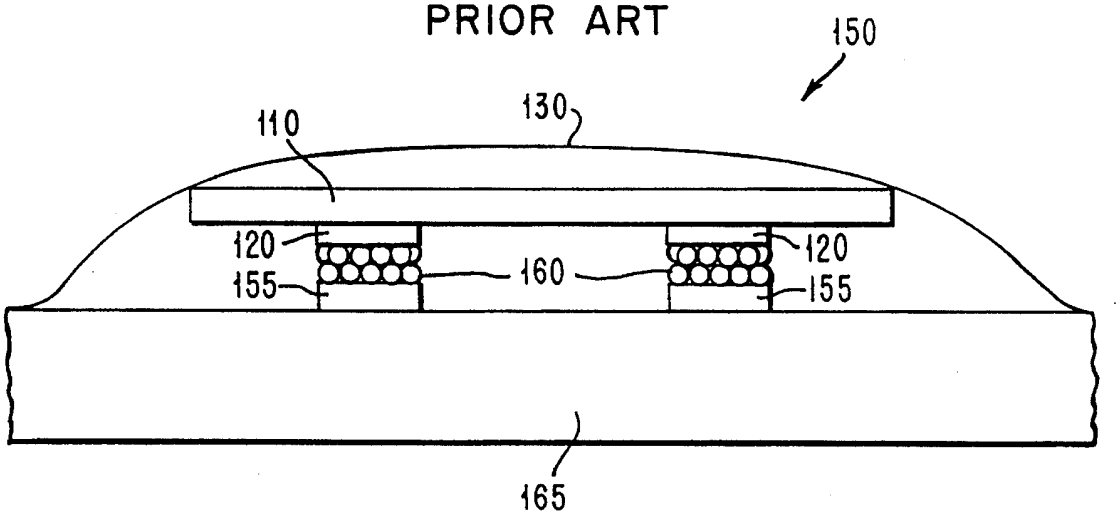


FIG. 2

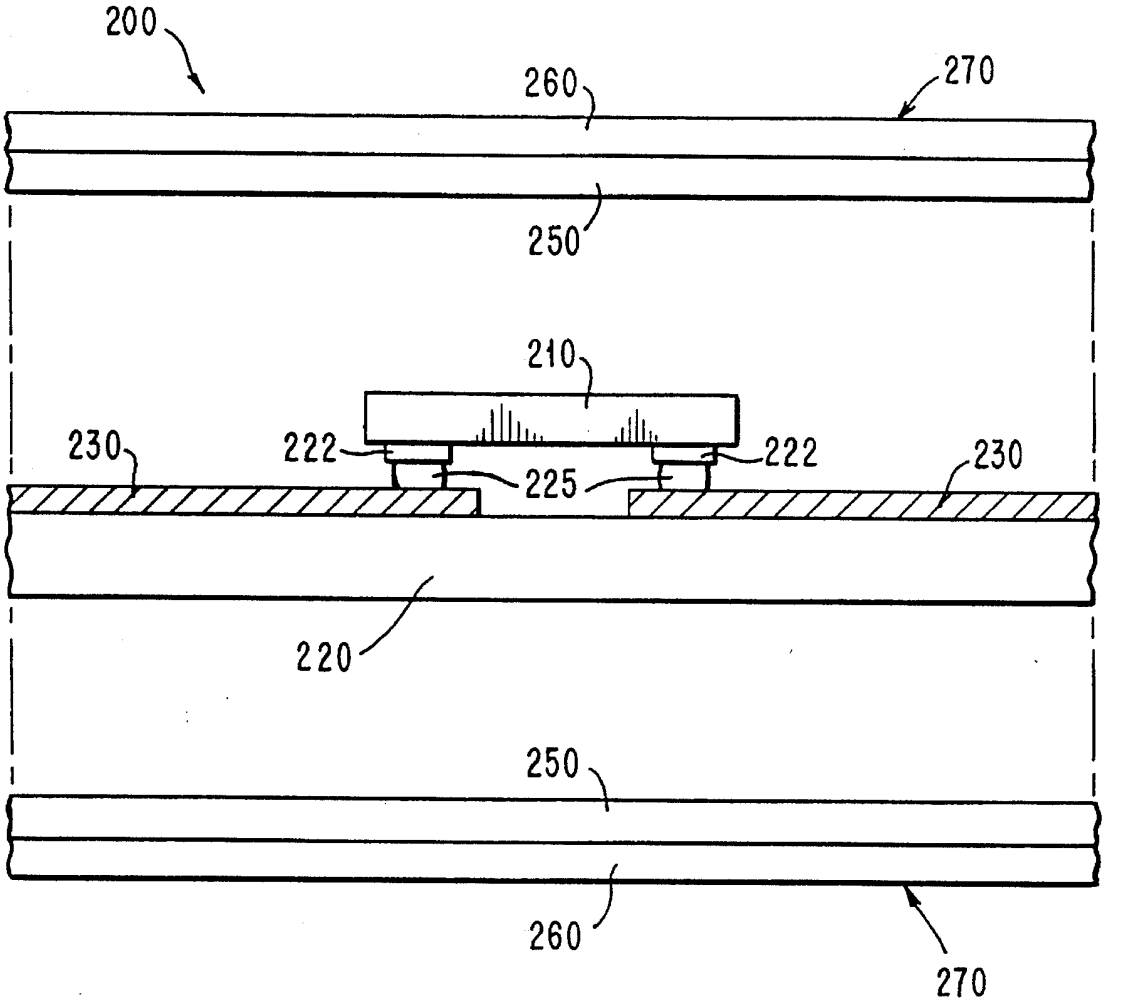


FIG. 3

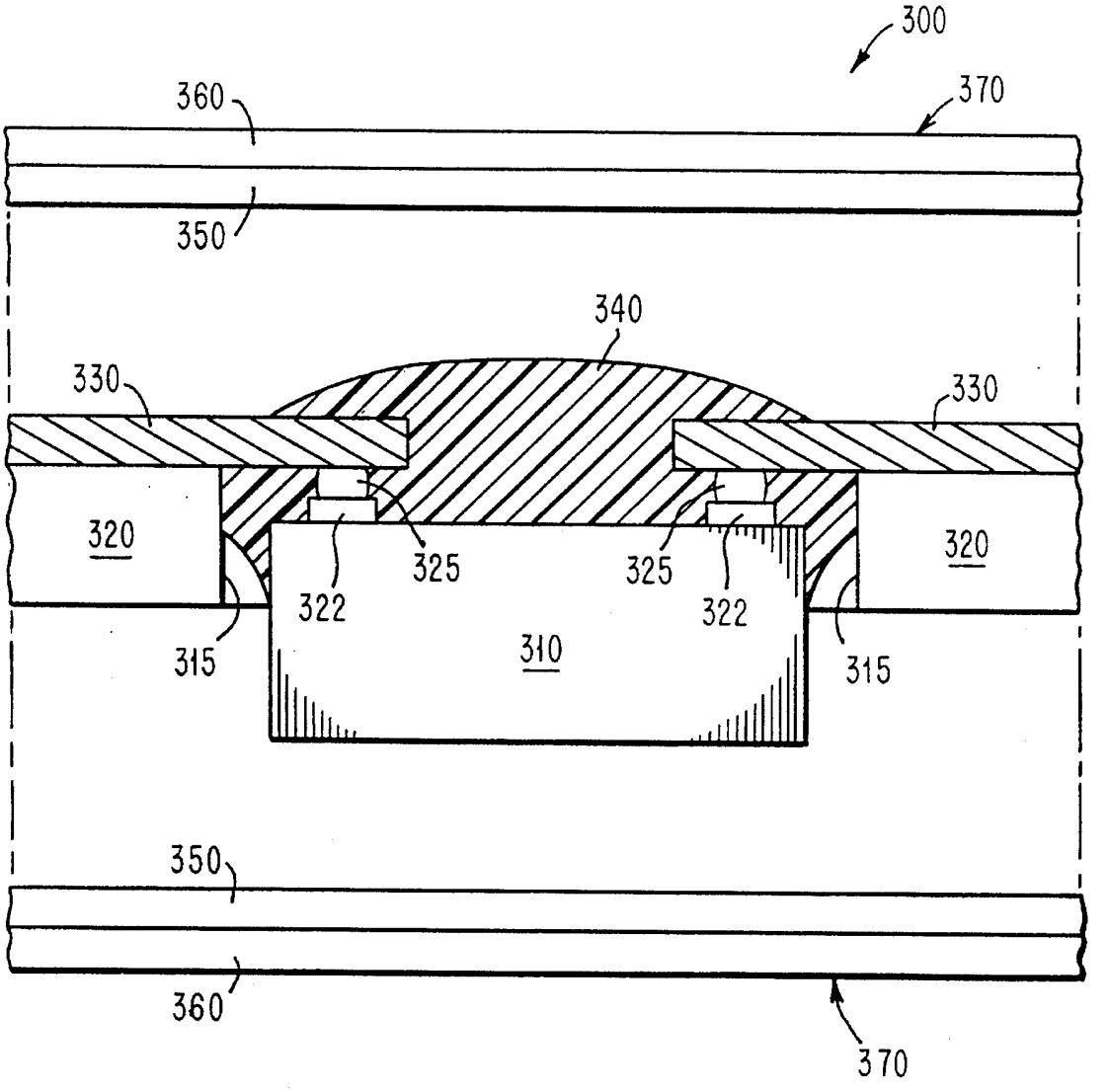

300
360
370
350
340
330
330
320
320
325
325
322
322
315
315
310
350
360
370

FIG. 4

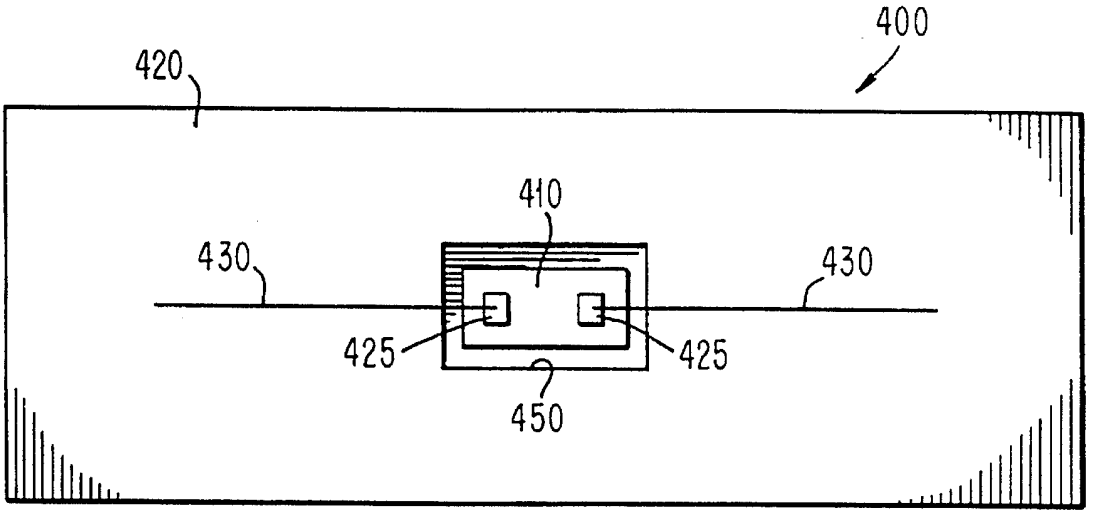

400
420
410
430
430
425
425
450

FIG. 5

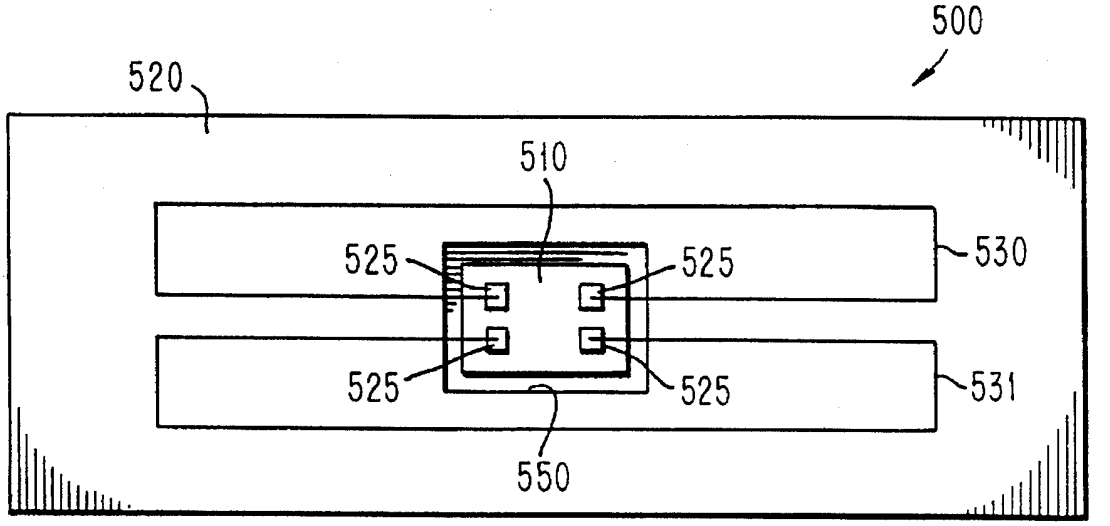

500
520
510
525
525
530
525
525
531
550

FIG. 6

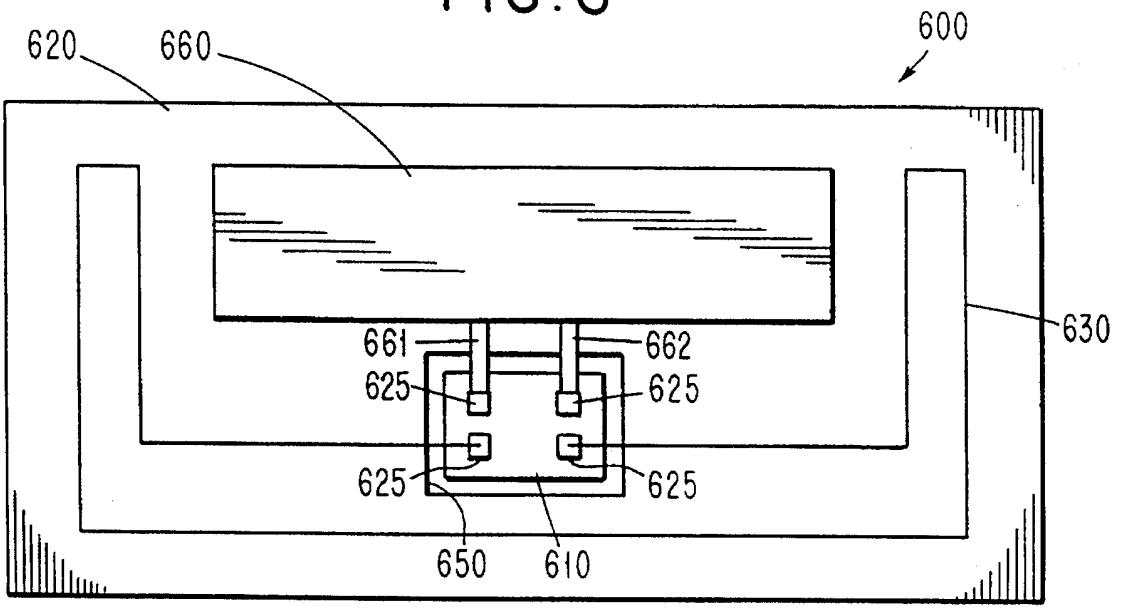


FIG. 7A PRIOR ART

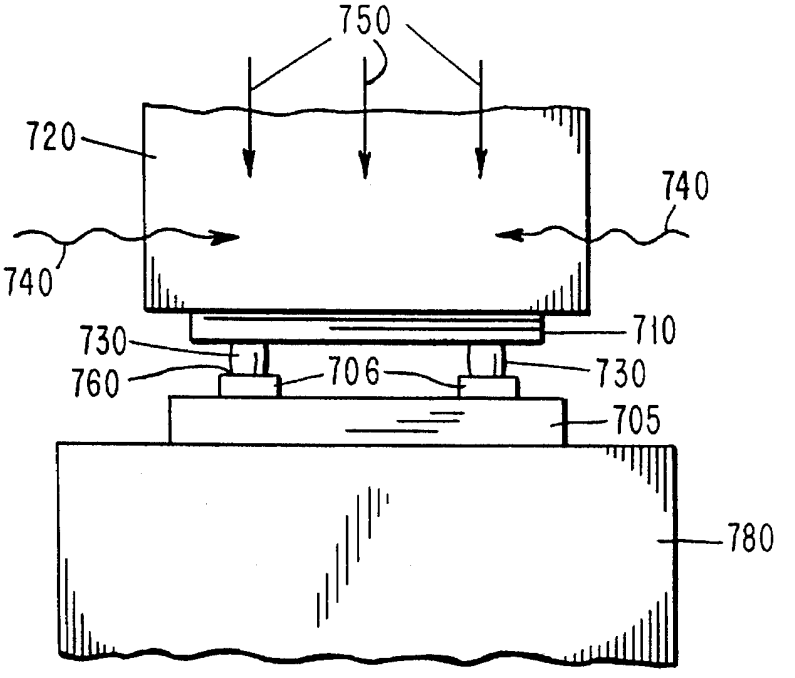


FIG. 7B
PRIOR ART

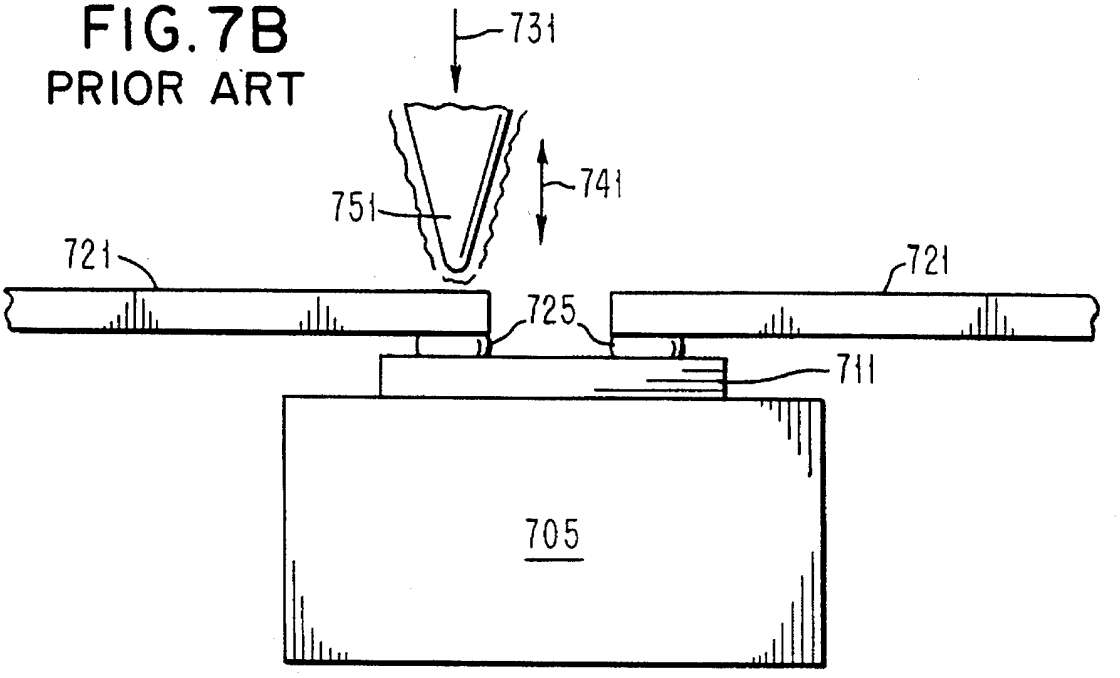


FIG. 7C
PRIOR ART

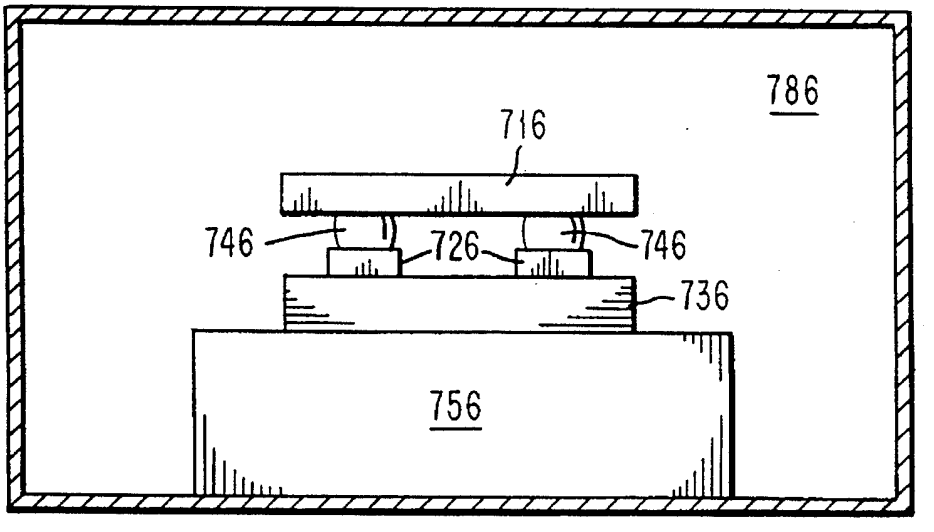


FIG. 7D
PRIOR ART

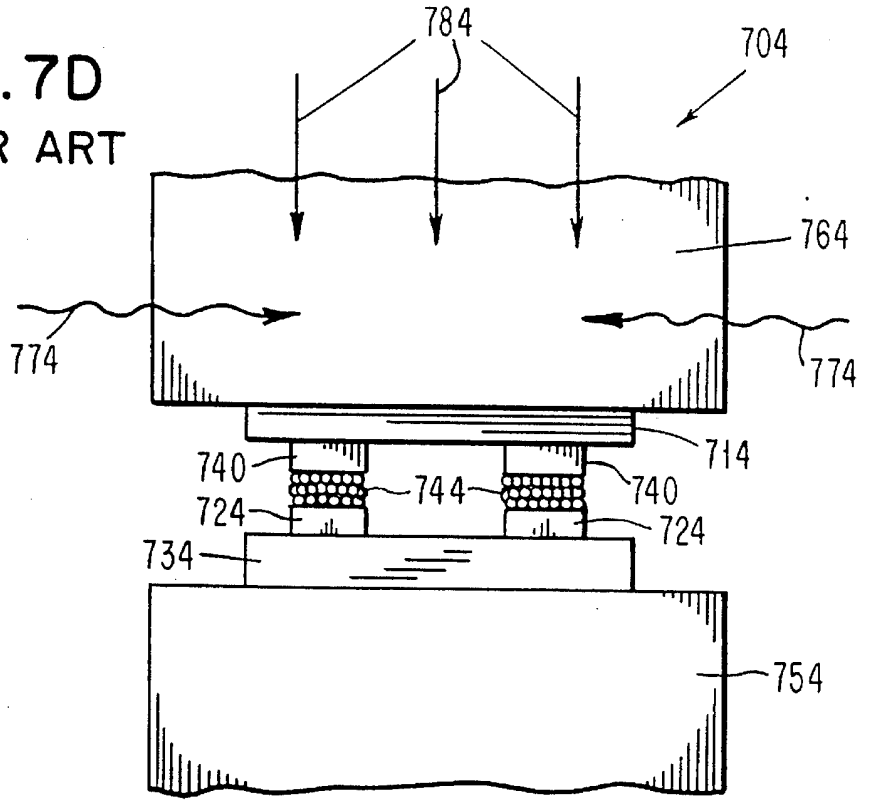


FIG. 7E
PRIOR ART

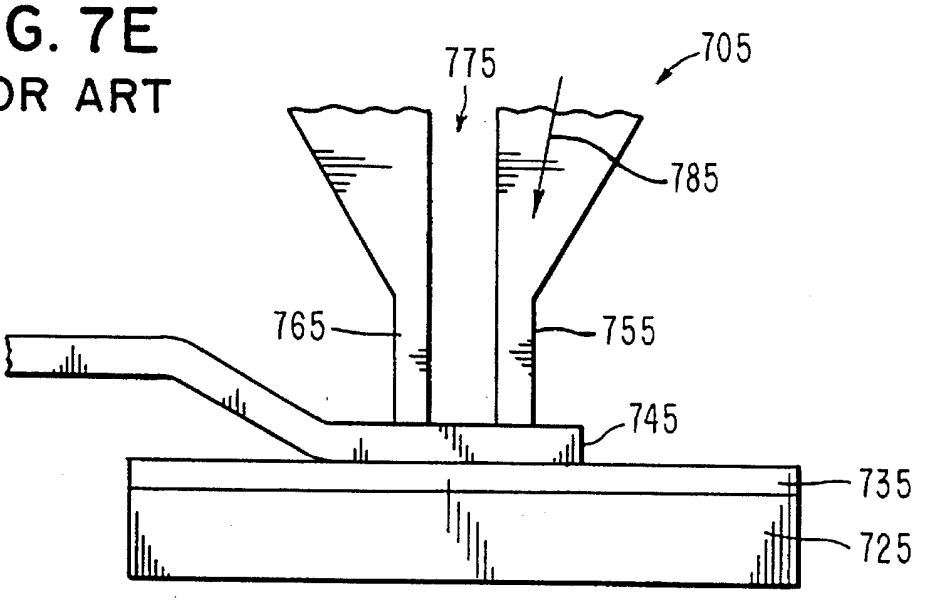


FIG. 8

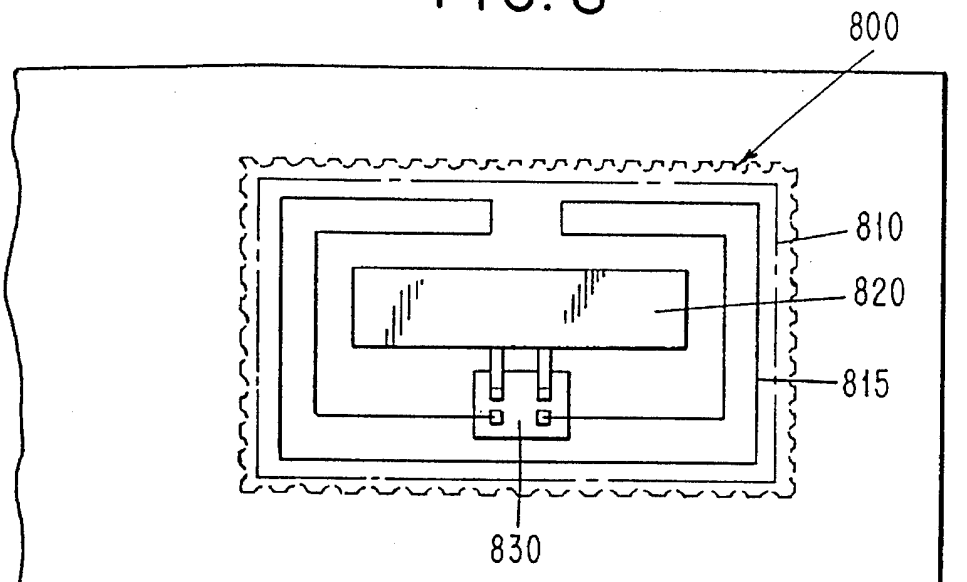


FIG. 8A

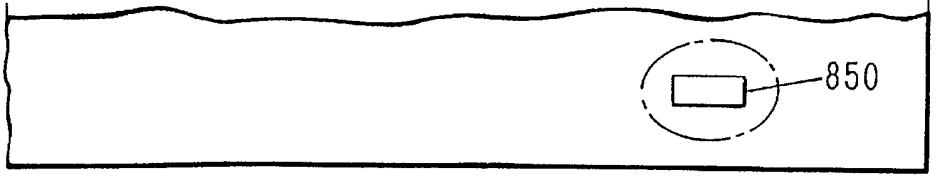


FIG. 9

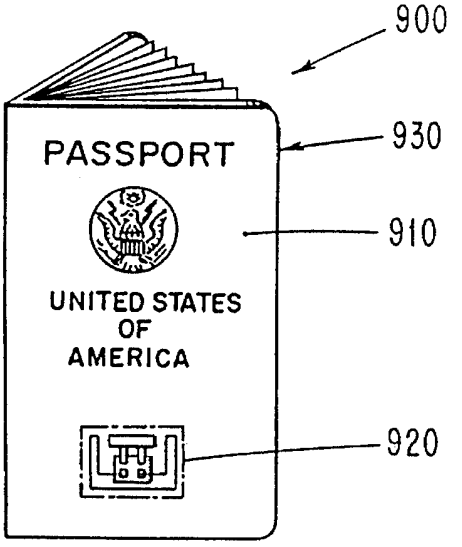


FIG. 10

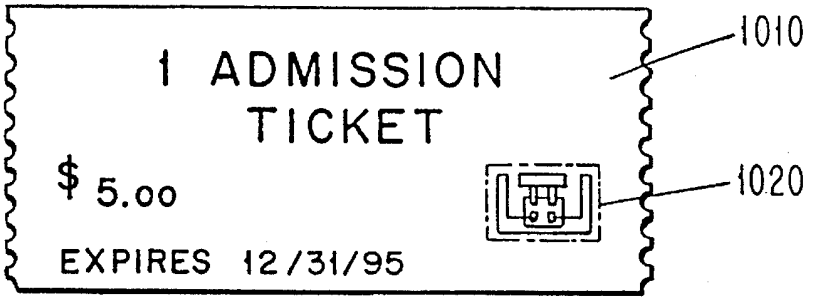

1 ADMISSION
TICKET
$ 5.00
EXPIRES 12/31/95
1010
1020

FIG. 11

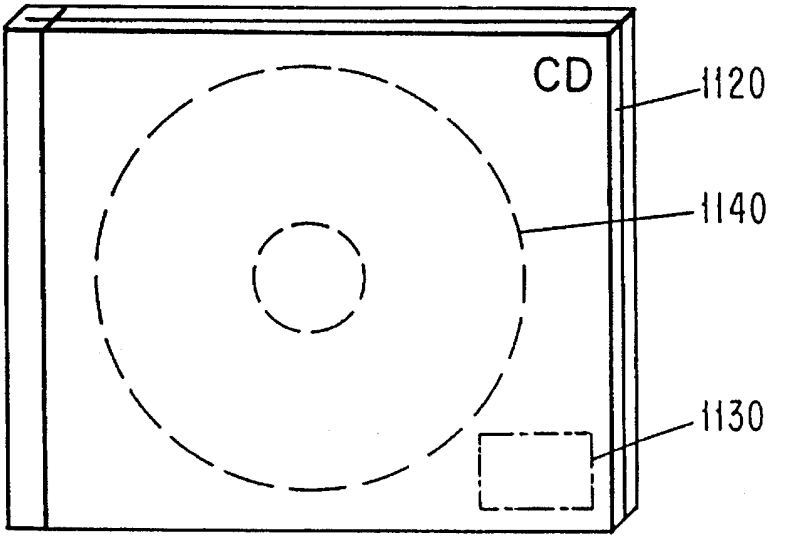

CD
1120
1140
1130

FIG. 12

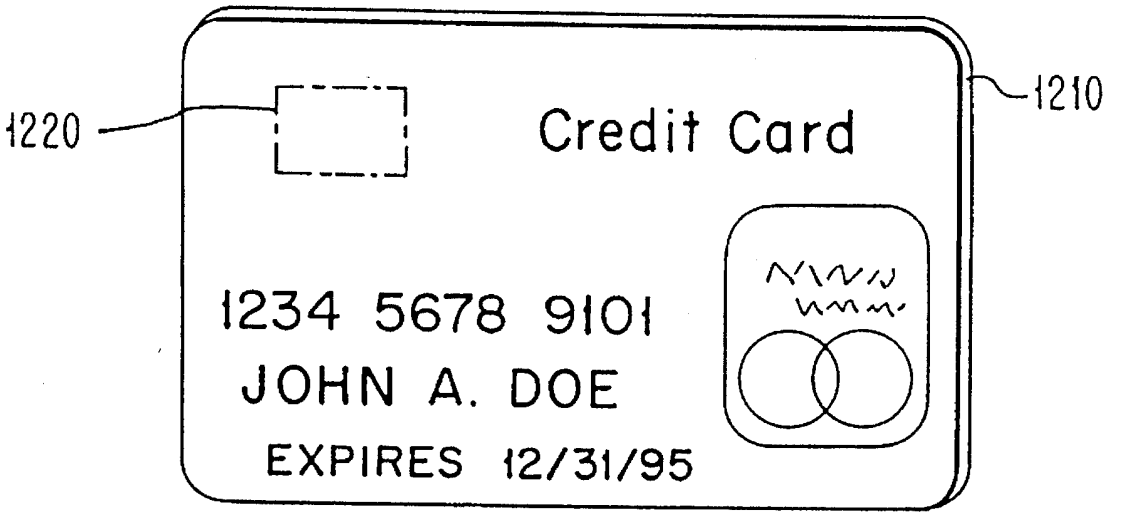
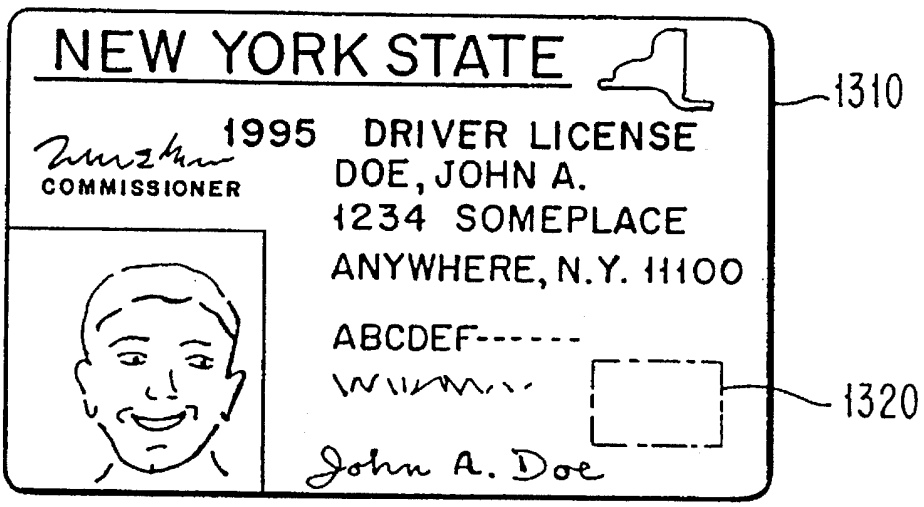

1220
Credit Card
1210
1234 5678 9101
JOHN A. DOE
EXPIRES 12/31/95

FIG. 13


NEW YORK STATE
1995 DRIVER LICENSE
COMMISSIONER
DOE, JOHN A.
1234 SOMEPLACE
ANYWHERE, N.Y. 11100
ABCDEF------
1310
1320
John A. Doe

RADIO FREQUENCY CIRCUIT AND MEMORY IN THIN FLEXIBLE PACKAGE

FIELD OF THE INVENTION

This invention relates to a radio frequency circuit and memory in a thin flexible package. More specifically, the invention relates to a thin flexible radio frequency circuit used as a radio frequency tag.

BACKGROUND OF THE INVENTION

Radio Frequency Identification (RF ID) is just one of many identification technologies for identifying objects. The heart of the RF ID system lies in an information carrying tag. The tag functions in response to a coded RF signal received from a base station. Typically, the tag reflects the incident RF carrier back to the base station. Information is transferred as the reflected signal is modulated by the tag according to its programmed information protocol.

The tag consists of a semiconductor chip having RF circuits, logic, and memory. The tag also has an antenna, often a collection of discrete components, capacitors and diodes, for example, a battery in the case of active tags, a substrate for mounting the components, interconnections between components, and a means of physical enclosure. One variety of tag, passive tags, has no battery. They derive their energy from the RF signal used to interrogate the tag. In general, RF ID tags are manufactured by mounting the individual elements to a circuit card. This is done by using either short wire bond connections or soldered connections between the board and the circuit elements: chip, capacitors, diodes, antenna. The circuit card may be of epoxy-fiberglass composition or ceramic. The antennas are generally loops of wire soldered to the circuit card or consist of metal etched or plated on a circuit card. The whole assembly may be enclosed in a plastic box or molded into a three dimensional plastic package.

While the application of RF ID technology is not as widespread as other ID technologies, bar code for example, RF ID is on its way to becoming a pervasive technology in some areas, notably vehicle identification.

Growth in RF ID has been inhibited by the high cost of tags, the bulkiness of most of the tags, and problems of tag sensitivity and range. A typical tag costs in the $5 to $10 range.

Companies have focused on niche applications. Some prior art is used to identify railway boxcars. These tags tend to be quite large and are made of discrete components on circuit boards mounted in solid, non-flexible casings. RF tags are now used in the automatic toll industry, e.g. on thruway and bridge tolls. RF tags are being tested for uses as contactless fare cards for buses. Employee identification badges and security badges have been produced. Animal identification tags are also commercially available as are RF ID systems for tracking components in manufacturing processes.

Tags exist that have the-length and width of a standard credit card. However, these cards typically are over 2.5 mm thick and have a non-flexible casing. Tags also exist that have a credit card size length and width but with bumps where circuit is placed that causes them to be too thick to fit in card reader machinery

While some electronic article surveillance (EAS), e.g. antitheft devices, are thin (0.3 mm) they typically contain limited amounts, (i.e., only one bit) of information. Some of these devices can be turned off once but cannot be reactivated.

FIG. 1A shows one prior art structure of a radio frequency tag **105**. The tag **105** has a chip **110** mounted on a substrate **115**. The chip **110** has contacts **120** that are connected to circuitry on the substrate **115** by wire bonds **125**. An encapsulation material **130** covers the chip for environmental protection. The thickness of this tag **105** is determined by the combined thicknesses of the chip components. Typically, substrates in these tags are at least 10 mils, 0.25 mm, in thickness, the chip **110** along with the high loop **122** of the bond vary from 20 to 40 mils, 0.5 to 1 mm, in thickness and the encapsulation **130** is about 10 mils, 0.25 mm in thickness. As a result, tags **105** of this structure vary from a minimum of 40 to 60 mils, 1 to 1.5 mm, in thickness. This structure is too thick for many potential tag applications.

FIG. 1B shows another prior art structure **150** showing a chip **110** with the chip contacts **120** connected to circuitry contacts **155** with conducting adhesive **160**. The substrate **165** of this structure **150** is typically made as a FR4/printed circuit (thickness 40 to 60 mils, 1 to 1.5 mm) or flexible substrate (10 mils,0.25 mm). The chip **110** and adhesive **160** add another 20 to 40 mils, 0.5 to 1 mm, to the thickness and the encapsulation **130** adds still another 10 to 20, 0.25 to 0.5 mm mils in structure **150** thickness. This structure therefore can vary in thickness from 80 to 130 mils, 2 to 3.5 mm, making it thicker than the structure in FIG. 1A.

Other thick structures are known in the art. These include quad flat pak (QFP) and/or small outline pak (SOP) as components. Structures made with these components are at least 1 mm thick and usually 2 to 3 mm thick.

PROBLEMS WITH THE PRIOR ART

Prior art teaches that there is a long felt need to manufacture thin RF ID tags on flexible substrates. However, while the goal of a thin flexible tag is desired, the prior art has failed to reach the goal. One prior art reference discloses a tag that is 1.5 to 2.0 mm thick. This tag thickness limits the applications of this tag. For example, it is far thicker than the International Organization for Standardization (ISO) standard credit card thickness of 0.76 mm and therefore could not be used in a credit card to be inserted into a credit card reader.

The prior art has failed to produce a thin tag because: care is not been taken to make each of the elements thin; elements are stacked one upon the next; and the antenna and connecting conductors require more than one plane of electrical wiring, ie. the designs use cross-overs for completing interconnections. As elements are stacked and layers are added the package grows thicker and flexibility is lost.

Another prior art reference discloses a package with a total thickness of 0.8 mm. This is still greater than the ISO standard credit card thickness of 0.76 mm. Furthermore, while thin elements are disclosed, no care is taken to use flexible materials throughout. The components are mounted on a hard circuit card and encapsulated in plastic. (Hard means can not be torn easily by hand.) The result a is rigid package. The prior art has not shown the use of thin flexible laminate covering materials for the packages. The results are that the packages are thick, and inflexible.

OBJECTS OF THE INVENTION

An object of this invention is an improved thin radio frequency tagging apparatus.

An object of the invention is a flexible radio frequency tag apparatus with a thin flexible protective lamination.

An object of the invention is a flexible radio frequency tag apparatus that may fit within the thickness limit of an ISO standard credit card, a passport cover, a postage stamp, an anti-theft device, or an admission ticket.

SUMMARY OF THE INVENTION

The present invention is a novel radio frequency (RF) tag that comprises a semiconductor circuit that has logic, memory, and radio frequency circuits. The semiconductor is mounted on a substrate and is capable of receiving a RF signal through an antenna that is electrically connected to the semiconductor through connections on the semiconductor. The present invention is a novel structure of a radio frequency tag design that is thin and flexible. The tag has the antenna and all interconnections placed on a single plane of wiring without crossovers. The elements of the package are placed adjacent to one another, i.e., they are not stacked. Elements of the package, the substrate, antenna, and laminated covers, are flexible. The elements are all thin such that the total package thickness including covers does not exceed that of an ISO standard credit card. The resulting tag package, comprised of thin, flexible components arranged and connected in a novel way, is also thin and flexible. Accordingly, this enables a novel range of applications that include: RF ID tagging of credit cards, passports, admission tickets, and postage stamps.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1**, comprising FIGS. **1**A and **1**B, is a drawing showing the cross section view of two typical embodiments in the prior art.

FIG. **2** is drawing showing a cross section of one preferred embodiment of the present thin RF ID tag.

FIG. **3** is drawing showing a cross section of one preferred embodiment of the present thin RF ID tag with an aperture in the substrate.

FIG. **4** is a top view of the thin tag showing a dipole antenna.

FIG. **5** is a top view of a thin tag having more than one folded dipole antennas.

FIG. **6** is a top view of a thin tag having a battery included in the circuit.

FIG. **7** comprises FIGS. **7**A–**7**E which are cross sections of prior art chip bonds to substrates by means of thermo-compression bonding (FIG. **7**A), ultrasonic bonding (FIG. **7**B), C4 solder bonding (FIG. **7**C), conducting adhesive bonding (FIG. **7**D), and spot welding (FIG. **7**E).

FIG. **8** shows a thin tag used as a postage stamp.

FIG. **8**A shows a thin tag enclosed in a parcel membrane or in the wall of an envelope.

FIG. **9** shows a thin tag placed in the cover of a passport using a resonant loop antenna.

FIG. **10** shows a thin tag used on an admission ticket.

FIG. **11** shows a thin tag used as an antitheft device.

FIG. **12** shows a thin tag placed inside a credit card.

FIG. **13** shows a thin tag placed inside a license.

DETAILED DESCRIPTION OF THE INVENTION

FIG. **2** shows a side view of a novel RF ID tag **200**. The chip **210** is located on a flexible substrate **220**. The chip **210** with bumps **225** on contacts **222** is bonded to an antenna **230** contained on the substrate **220**. The package is sealed by thin flexible laminations **270** consisting of a hot-melt adhesive **250** such as EVA on the inside and an outer coating **260** of a tough polymeric material on the outside.

The antenna is manufactured as an integral part of the substrate. It will consist of thin, typically 25 to 35 micron thick copper lines which have either been etched onto a copper/organic laminate or plated on the organic surface. The thinness of the copper maintains the flexibility of the substrate. Typical materials used are polyester or polyimide for the organic and electroplated or rolled annealed copper. The copper may be gold or tin plated to facilitate bonding. The chip is connected to the antenna lines by means of bumps on the chip, either plated gold bumps for thermo-compression bonding or C4 solder bumps for solder bonding are preferred. The bumps **225** then become the connecting lines. Since they are only on the order of 25 microns or so they will not degrade electrical performance by introducing unwanted inductance into the circuit. The novel design has a single metal layer with no vias (between-plane connectors through a dielectric layer) in the flexible continuous film. By using only one level of metal to produce the antenna and interconnections, the package is kept thin. Further novelty of the invention includes arranging the components (chip and antenna and possibly a battery) in adjacent proximity to one another. This means that the components are close (i.e., not stacked). In a more preferred embodiment the closeness is insured because the chip **210** is bonded directly to the antenna **230** without the use of crossovers in the circuit. This is accomplished by using either a dipole, loop or folded dipole antenna that is resonant rather than using a multiloop antenna which requires cross-overs for connection. Thus all of the wiring is placed in a single plane. Keeping the antenna adjacent to the chip, avoiding cross-overs and stacking, also contributes to keeping the package thin.

To maintain the thinness of the package, the chill is made to be 225 to 375 microns thick by thinning. In general, semiconductors are manufactured on thick wafers, up to 1 mm thick. Thinning may be done by polishing or backgrinding of the wafer after manufacture. All elements and bonds are very thin. The elements are preferably: the chip (and battery if used) are 10 to 12 mils (250 to 300 microns) thick or thinner; the bonding structures are 2 mils (50 um) or less; laminating materials 2 to 4 mils (50 to 125 um) per side; to produce total thickness preferably of about 20 mils (500 um) or less but in any case less than 30 mils (750 um). Bonding mechanisms do not add to thickness of the tag as would techniques like wirebonding.

Although not required in one preferred embodiment, a unique flexible covering material **270** may be laminated upon one or both sides of the package. In another preferred embodiment, the material consists of two layers (**250**, **260**). A soft copolymer such as ethyl-vinyl-acetate is located on the inside **250** surface of the cover. Tough polyester is located on the outside **260** surface. This combination provides environmental protection while maintaining the flexibility of the package. Typical thicknesses of the covers range from 50 to 125 microns. Alternately, a single layer of laminate such as polyethylene, polyester, mylar or polyimide may be used for covering.

FIG. **3** shows a side view of a unique RF ID tag **300**. The chip **310** with contacts **322** and bumps **325** is bonded to antenna **330** thru window **315** in substrate **320**. In a more preferred embodiment, encapsulant **340** is used to protect the chip **310**, the bonds **325** on contacts **322**, connected to antenna **330** located in window **315** between substrate **320** from environmental exposure. In a still more preferred

embodiment, the package is sealed by thin flexible laminations **370** consisting of hot melt adhesive **350** such as EVA, phenolic butyral, or silicone adhesive on the inside and an outer coating **360** of a tough polymeric material (such as polyester, mylar, polyimide, and polyethylene) on the outside. In an alternative preferred embodiment, layer **370** comprises a single layer of organic material.

In order to further reduce the thickness of the package, the substrate is manufactured with a window allowing the insertion of the chip into the window. Thus, the thickness of the substrate is not added to the thickness of the chip. The window is produced in organic materials, polyimide or polyester by either etching or punching. In addition, the window may be used to allow the coating of the chip with a thin layer of encapsulation material. Hysol epoxy **4510** is one such material. The encapsulant does not add substantially to the total package thickness, adding perhaps 50 microns, but does provide additional environmental protection for the chip. Opaque materials in the encapsulant protect light sensitive circuits on the chip. In this embodiment, the antenna and the center of the chip can be coplanar.

FIG. **4** shows a top view of the thin RF ID tag **400**. The chip **410** is located within a window **450** placed in a flexible substrate **420**. The chip **410** has contacts **425** which are connected to a antenna **430** contained on the substrate.

FIG. **5** shows a top view of the thin RF ID tag **500**. The chip **510** placed in the window **550** has contacts **525** which are connected to more than one folded dipole antenna **530** and **531** contained on the substrate.

FIG. **6** shows a top view of the thin tag **600**. The semiconductor chip **610** is connected to a folded dipole antenna **630** by means of contacts **625**. The antenna is contained in the substrate **620** as described above. A thin battery **660** is connected to the chip **610** by leads **661** and **662** bonded at contacts **625**.

The battery has short connecting lines **661** and **662** providing electrical continuity between the battery and the chip. The battery is placed adjacent to the chip, not stacked upon the chip. The battery thickness of about 0.25 mm keeps the battery flexible. The antenna is designed such that it is also adjacent to the battery. There is no overlap. The wiring is kept in one plane and all of the elements (chip, battery, antenna) are coplanar; there is no stacking. As a result, the package is thin and flexible.

The bonding method for attaching batteries to prior art radio frequency tags include some of the techniques described below, i.e., soldering, conducting adhesive; and wire bonding. In addition, spot welding may be used. In spot welding, shown below in FIG. **7E**, the battery connection pads are pressed to contacts on the substrate while a low-voltage high-current pulse bonds the two metals together.

In one preferred embodiment, the metallurgies on the battery, chip, and substrate are such that the battery attaching mechanism is consistent with the method and mechanism of the chip attachment. For example, use of tin plating on the substrate to enable chip bonding may preclude use of conductive adhesive to attach the battery but might allow use of gold plating to enable attaching of both.

A more preferred embodiment used to make a thin flexible rugged package uses robust chip attach techniques such as thermocompression (TC) bonding used in TAB (tape automated bonding) technology. Using TC bonding for the chip and spot welding for the battery is a novel combination of bonding techniques that enables attachment of the battery to a flexible substrate **620**. In one preferred embodiment, the substrate is a TAB polyimide or polyester.

FIG. **7** shows different types of bonding available in the prior art to attach chips to circuitry that are on the substrate when producing an RF tag. These include thermocompression bonding, ultrasonic single point bonding, soldering, and conductive adhesive.

In FIG. **7A**, using thermocompression bonding, suitable metal surfaces are brought into contact with pressure **750** and heat **740** applied by thermode **720** to form a metal-to-metal bond **760** usually gold bumps **730** on chip **710** to gold-plated leads **706** on substrate **705** which rests on lower thermode **780**. Many leads are bonded at once (gang bonding). This is used extensively for reel-to-reel TAB (tape automated bonding).

FIG. **7B** shows ultrasonic singlepoint bonding a variation on thermocompression bonding for TAB where some ultrasonic energy is substituted for some pressure. One bond is done at a time. This bonding type also requires gold-to-gold metallurgy. Bonding tip **751** applies pressure **731** and ultrasonic energy **741** while pressing lead **721** to bump **725** on chip **711** resting on lower support **705**.

FIG. **7C** shows soldering or C4 solderbonding where small lead/tin solder bumps **746** are used as the connecting medium between chip **716** and pads **726** on substrate **736**. The reflow is carried out while the substrate is carried on platform **756** through oven **786**. This usually requires the application of solder flux for reflow of the solder at elevated temperature.

FIG. **7D** shows conducting adhesive bonding where a metal-filled adhesive **744** is applied to form the connecting medium between chip pads **740** on chip **714** and the substrate pads **724** on the substrate **734**. Heat **774** and pressure **784** are applied by pressing between thermodes **764** and **754**.

FIG. **7E** shows spot welding where welding tips **755** and **765** separated by gap **775** are pressed to conductor **745** held in contact with conductor **735** placed on insulating substrate **725**. Current **785** heats the welding tips **755** and **765** to make the bond.

FIG. **8** shows an RF postage stamp **800** containing a thin RF tag **810** which consists of antenna **815**, battery **820**, and chip **830** affixed to envelope or package **840**. This tag **810** can be any of the embodiments described above. In this application, the cover (typically **270** of FIG. **2** and **370** of FIG. **3**) for the tag is the paper of the stamp. Adhesives, such as acrylics, are used to sandwich the tag between thin paper. These adhesives would correspond to the layer **250** in FIG. **2** and **350** in FIG. **3**. The top surface (of one side **270**, **370**) can be printed with the appropriate graphics while the bottom surface has a pressure sensitive adhesive (of the other side **270**, **370** in the case of a tag laminated on two sides), also acrylic, to bond the stamp to a package or letter envelope. The RF tag would contain information about mailing used to track a letter or parcel on which the stamp is placed. Alternatively, the RF tag **850** could be enclosed in the parcel membrane or in the wall of the envelop **840**. In another, embodiment the RF tag could be placed within the parcel or envelop.

FIG. **9** shows the thin RF tag **920** embedded in the cover **910** of passport **930** to form an RF passport **900**. Here the tag is sandwiched between the paper covers of the passport. The tag can have an environmental laminate(s) (**270**, **370**) as described above or alternatively, the passport cover can be used as the tag laminate(s) (**270**, **370**). The tag contains in its memory information on the identity of the passport owner, visas, dates of entry, restrictions, or any other desirable information. The information may be in encrypted form for added security. The encryption "key" would be a software

code that is held and used solely by the agency issuing the passport. The deencryption key may be made public so that anyone (with a public deencryption key) can read information in the memory of the tag but only the agency having the encryption key can write information to the tag.

FIG. **10** shows admission ticket **1010** containing RF tag **1020**. The tag is again enclosed between paper covers or other laminates. The ticket may be a simple admission ticket or entitlement such as an airline ticket or a food stamp. However, the tagged ticket may also serve as a tracking device.

FIG. **11** shows a CD **1140** enclosed in box **1120** with an RF ID antitheft tag **1130** affixed to the box **1120**. The tag serves as both a barcode replacement, inventory device, point of sale device, and as an antitheft device. Information on product variety, price, date of manufacture and sale may be carried by the tag. Additional bits of information in the memory of the circuit may be changed at the time of sale to indicated that the item may be taken from the store.

FIG. **12** shows ISO standard credit card **1210** containing an RF tag **1220**. The credit card may serve as an ATM card, frequent flyer card, library card, phone card, employee ID, medical ID card, gasoline credit card or any credit or debit card. The covers (laminates **270, 370**) of the tag could be the covers of the credit card, preferably PVC laminations. The core of the credit card, 0.5 mm thick, has a window placed in it at the time of manufacture. The 0.5 mm thick tag package is placed in the window and then sealed into the card. The resulting credit card, including the tag, will not only have the length and width that meet the ISO standard, but the thickness as well.

In another embodiment of the present invention, shown in FIG. **13**, the RF tag **1320** is place within a vehicular drivers license **1310** in the same manner as described above. This allows information on the RF tag to be used for personal identification, driving record, organ donor information, restrictions, proof of identity and age, etc. The information can be encrypted for security purposes.

We claim:

1. A thin flexible electronic radio frequency tag circuit comprising;

a. an insulating, flexible substrate;

b. an antenna that is an integral part of the substrate and that has terminals;

c. a circuit chip having a modulator circuit, a logic circuit, a memory circuit, and chip connectors and being on the substrate in adjacent proximity to the antenna;

d. one or more connecting lines between the antenna terminals and the chip connectors, the connecting lines being coplanar with the antenna and antenna terminals.

2. A circuit, as in claim **1**, wherein the substrate is organic.

3. A circuit, as in claim **2**, wherein the substrate is polyimide.

4. A circuit, as in claim **2**, wherein the substrate is polyester.

5. A circuit, as in claim **1**, wherein the connecting lines are bonded to the chip connectors using any of the bonding types including thermal compression, single point bonding, C4 bonding, and conductive adhesive.

6. A circuit, as in claim **1**, wherein the substrate has an aperture into which the chip is placed.

7. A circuit, as in claim **1**, wherein the chip is covered by an encapsulant.

8. A circuit, as in claim **7**, wherein the encapsulant is opaque.

9. A circuit, as in claim **7**, wherein an organic cover surrounds the chip, the encapsulant, the substrate, and the antenna.

10. A circuit, as in claim **9**, wherein the organic cover is one of the materials including polyester, mylar, polyimide, and polyethylene.

11. A circuit, as in claim **1**, that is laminated by one or more layers.

12. A circuit, as in claim **11**, that is laminated by a two layer laminate comprising a hard outer layer and a adhesive inner layer.

13. A circuit, as in claim **12**, wherein the adhesive is one of the materials including ethyl vinyl acetate (EVA), phenolic butyral, and silicone adhesive.

14. A circuit, as in claim **11**, wherein the circuit is laminated on one side.

15. A circuit, as in claim **11**, wherein the circuit is laminated on two sides.

16. A circuit, as in claim **11**, wherein the circuit has at least one tag dimension that is less than 760 microns (30 mils).

17. A circuit, as in claim **16**, that is encapsulated as an International Organization for Standardization (ISO) standard credit card size package.

18. A circuit, as in claim **1**, wherein the antenna is a resonant antenna and is any one of the following structures including folded dipole, dipole, and loop.

19. A circuit, as in claim **1**, wherein a battery is also affixed to the substrate in adjacent proximity to the antenna and chip and is connected by one or more battery connecting lines to two or more chip battery contacts wherein the battery connecting lines and the battery contacts are coplanar with the antenna and connecting lines.

20. A circuit, as in claim **19**, wherein the battery contacts are connected to the battery connecting lines by any of the bonding types including spot welding, soldering, thermocompression bonding, and conducting adhesive.

21. A circuit, as in claim **19**, wherein the battery contacts are connected by spot welding and the chip contacts are connected to the antenna by thermocompression bonding.

22. A circuit, as in claim **1**, wherein the chip has at least one chip dimension less than 300 microns (12 mils), the antenna has at least one antenna dimension less than 35 microns (1.4 mils), and the substrate has at least one substrate dimension less than 125 microns (5 mils) whereby the circuit has at least one circuit dimension less than 508 microns (20 mils).

23. A circuit, as in claim **22**, wherein the chip memory has information about mailing and the circuit is applied to a mailed letter or parcel.

24. A circuit, as in claim **23**, wherein the RF tag is enclosed within a stamp.

25. A circuit, as in claim **23**, wherein the RF tag is enclosed within the parcel or envelop membrane.

26. A circuit, as in claim **22**, wherein the tag is enclosed in a passport.

27. A circuit, as in claim **22**, wherein the tag is enclosed in an admission ticket.

28. A circuit, as in claim **22**, that is enclosed in an article and the tag has information to prevent theft.

29. A circuit, as in claim **22**, wherein the tag is enclosed in a drivers license.

* * * * *

EXHIBIT B

US005828318A

United States Patent [19]

Cesar

[11] Patent Number: 5,828,318

[45] Date of Patent: Oct. 27, 1998

[54] **SYSTEM AND METHOD FOR SELECTING A SUBSET OF AUTONOMOUS AND INDEPENDENT SLAVE ENTITIES**

[75] Inventor: **Christian Lenz Cesar**, Shrub Oak, N.Y.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **646,539**

[22] Filed: **May 8, 1996**

[51] **Int. Cl.**[6] **G08C 19/00**; G05B 23/02

[52] **U.S. Cl.** **340/825.69**; 340/825.06; 340/825.07; 340/825.08; 395/290; 395/291; 395/683

[58] **Field of Search** 340/895.69, 825.06, 340/825.07, 825.08, 825.12, 825.71, 571, 572; 395/290, 201, 683

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,970,824	7/1976	Walton et al.	235/61
4,636,950	1/1987	Caswell et al.	364/403
4,673,932	6/1987	Ekehian et al.	340/825
5,151,684	9/1992	Johnsen	340/572
5,231,273	7/1993	Caswell et al.	235/385
5,245,534	9/1993	Waterhouse et al.	364/404
5,407,050	4/1995	Takemoto et al.	194/205
5,410,315	4/1995	Huber	342/42
5,434,572	7/1995	Smith	342/44
5,489,908	2/1996	Orthmann et al.	342/42
5,590,339	12/1996	Chang	395/750

FOREIGN PATENT DOCUMENTS

0585132	3/1994	France	G01S 13/78

OTHER PUBLICATIONS

A. R. Grasso, Electronic Remote Identification—A Case Study: IEEE 1990, pp. 14–2.1–14–2.3.

Narain H. Gehani, Broadcasting Sequential Processess (BSP), 1984 IEEE Transactions on Software, Engineering, vol. SE–10, No. 4, 343–351, Jul. 1984.

TRIS (Texas Instruments Registration and Identification System, "Stationary Reading/Writing Unit Reference Manual", Manual No. RI–ACC–UM02–01, pp. 1 to 3.

Set Operations and the Laws of Set Theory, Enumerations in s pp. 45–53.

"Micron RFID Communications Protocol, Pre–Release Version 0.95, 1993 Micron Communications", Inc., pp. 1–71.

DuoProx Multiple Technology Proximity Card, Hughes Identification Devices, DP1330–L Series, pp. 1–2.

"Research Looks Into the Future", Ames Nelson, Think Magazine, Jul./Aug. 1994.

Primary Examiner—Michael Horabik

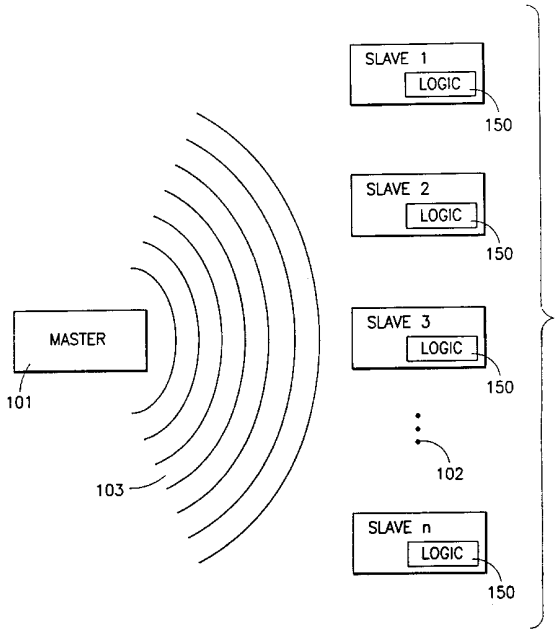
Assistant Examiner—Anthony A. Asongwed

Attorney, Agent, or Firm—Louis J. Percello; Ronald L. Drumheller

[57] **ABSTRACT**

A master entity is capable of broadcasting commands to a plurality of three-state-selection machine slaves. Transitions from one state to another are effected on instruction from commands in a sequence of commands broadcast from the master. Slaves move to another state when they satisfy a primitive condition specified in the command. By moving slaves among the three sets, a desired subset of slaves can be isolated in one of the sets. This desired subset of slaves then can be moved to one of the states that is unaffected by commands that cause the selection of other desirable subsets of slaves.

18 Claims, 35 Drawing Sheets



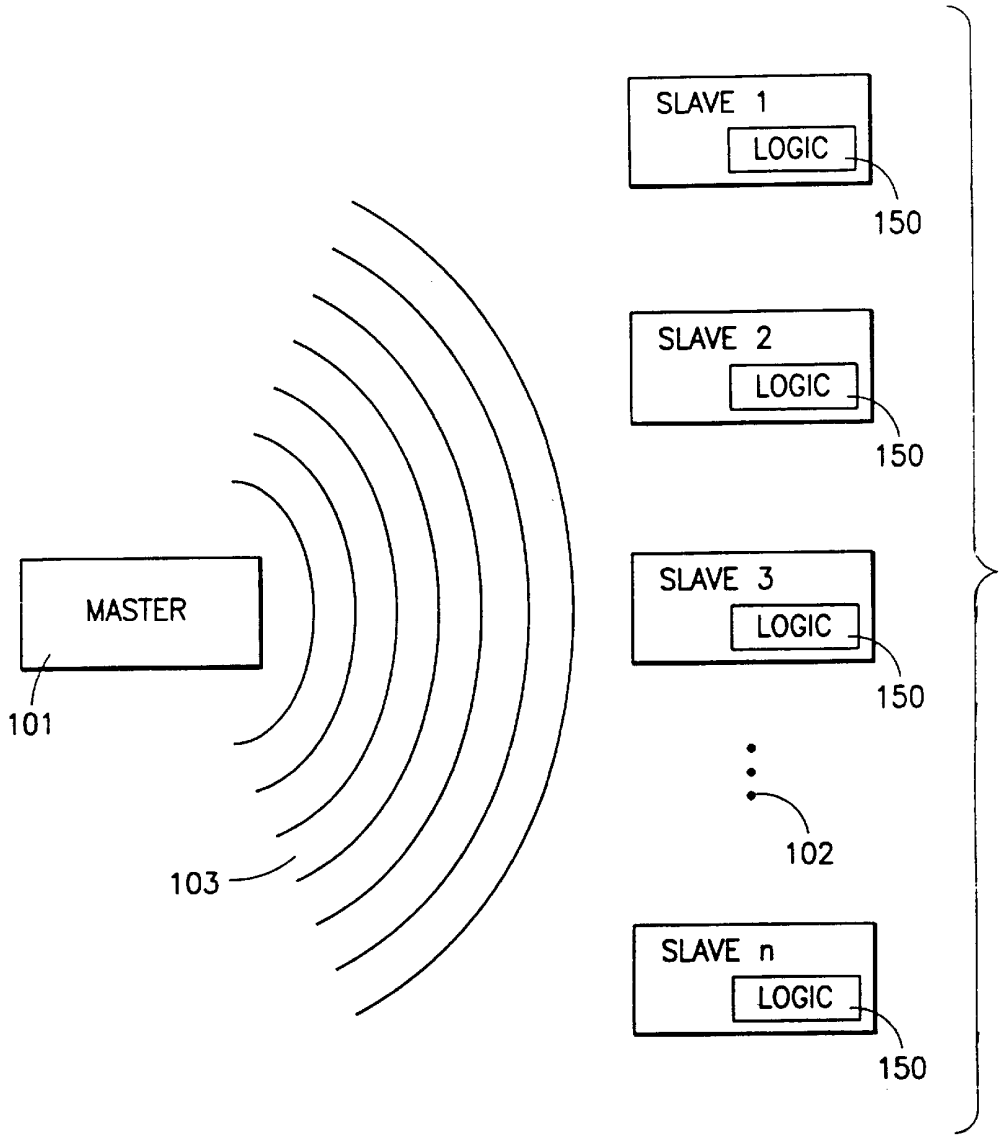

SLAVE 1
LOGIC
150
SLAVE 2
LOGIC
150
SLAVE 3
LOGIC
150
102
SLAVE n
LOGIC
150
MASTER
101
103

FIG. 1

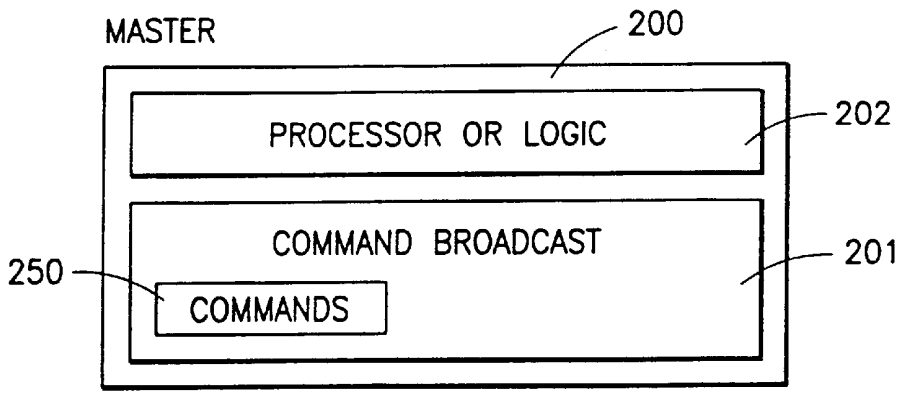

MASTER
200
PROCESSOR OR LOGIC
202
COMMAND BROADCAST
201
250
COMMANDS

FIG. 2

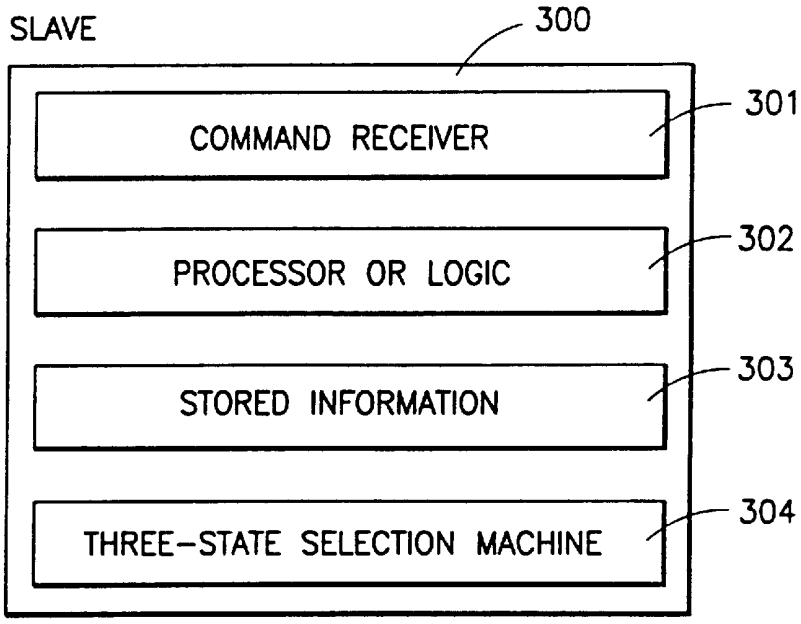

SLAVE
300
COMMAND RECEIVER
301
PROCESSOR OR LOGIC
302
STORED INFORMATION
303
THREE-STATE SELECTION MACHINE
304

FIG. 3

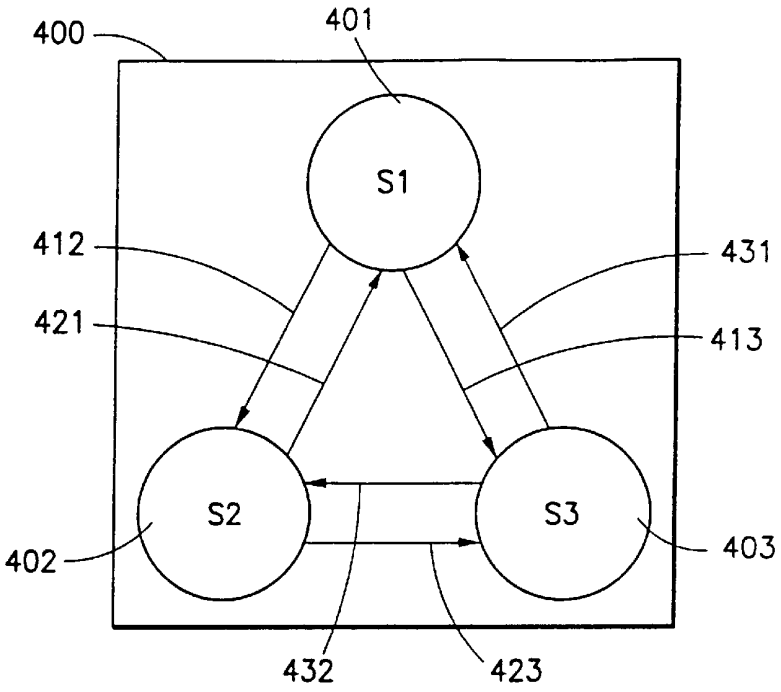

400
401
S1
412
421
431
413
S2
S3
402
403
432
423

FIG. 4

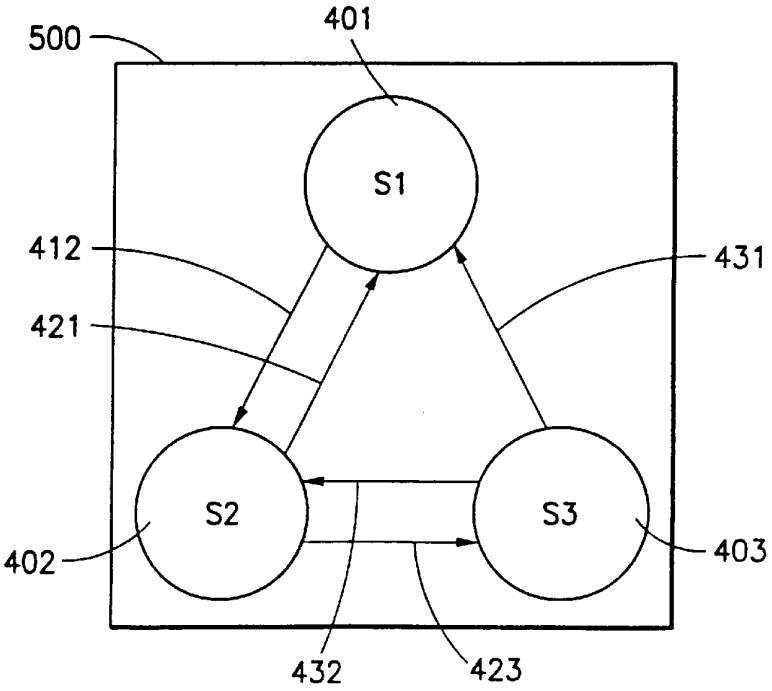

500
401
S1
412
421
431
S2
S3
402
403
432
423

FIG. 5

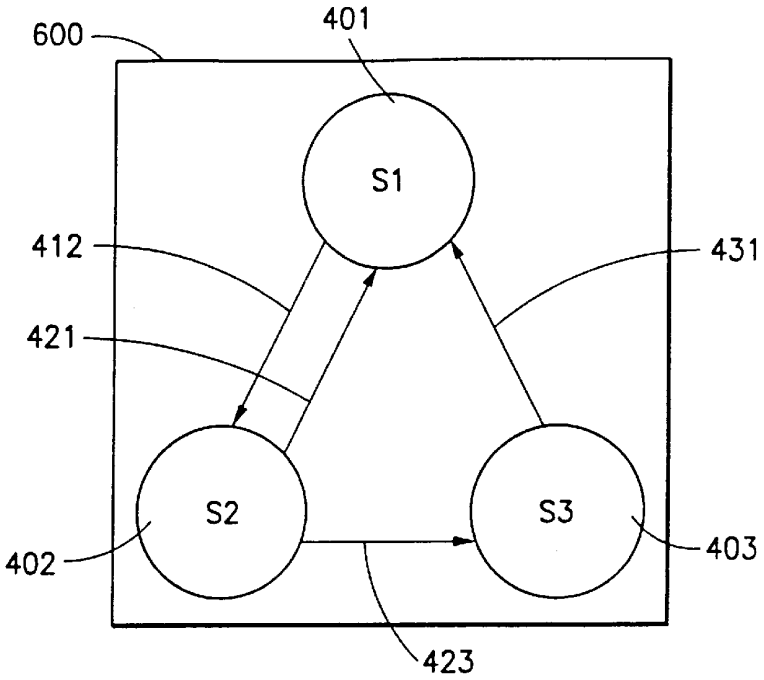

600
401
S1
412
431
421
S2
S3
402
403
423

FIG. 6

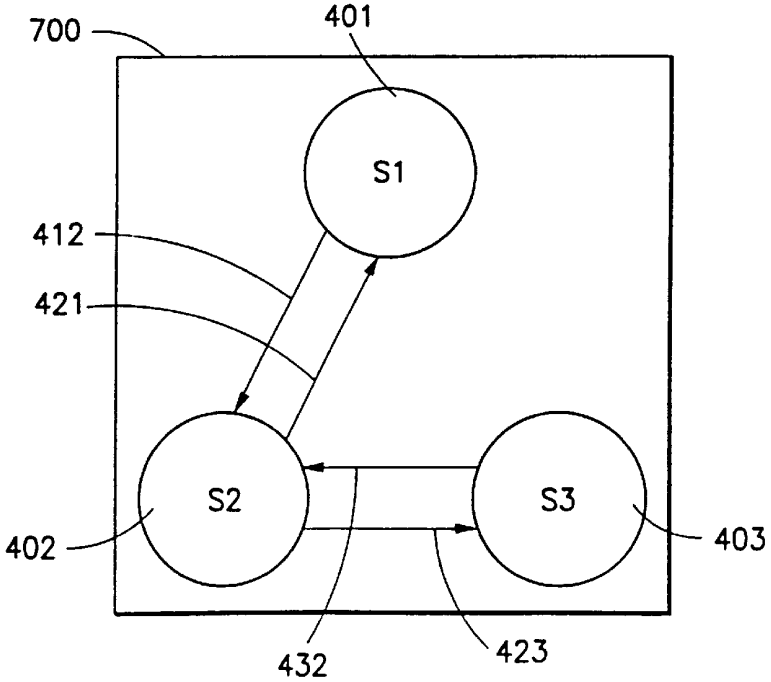

700
401
S1
412
421
S2
S3
402
403
432
423

FIG. 7

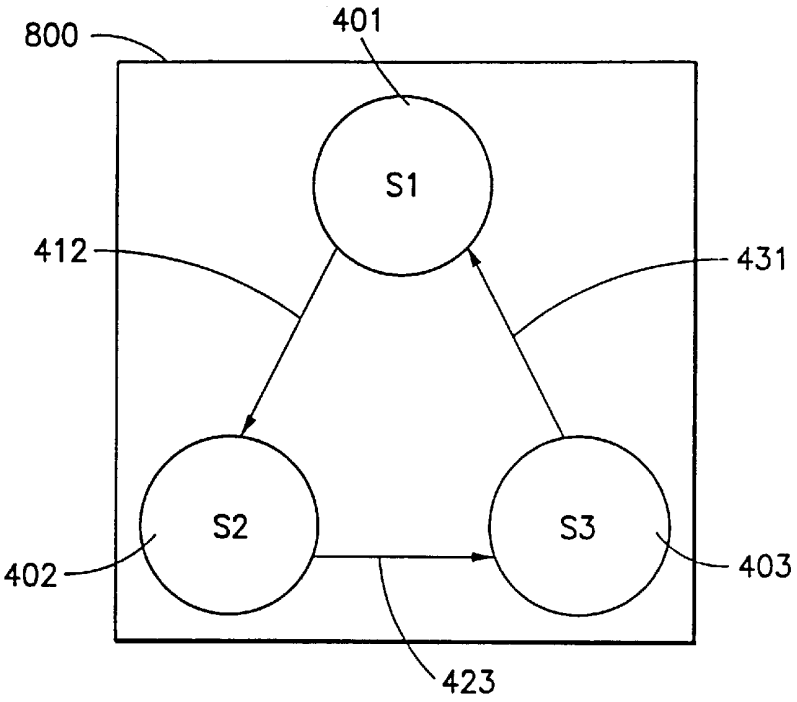

800
401
S1
412
431
S2
S3
402
403
423

FIG. 8

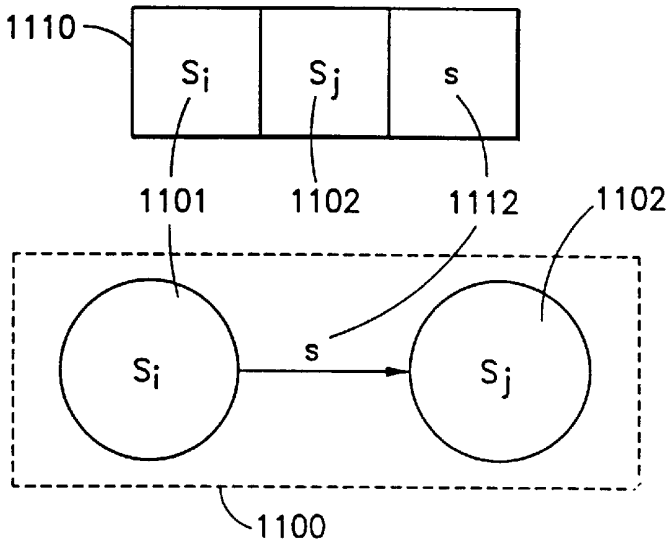

1110
Si
Sj
s
1101
1102
1112
1102
Si
s
Sj
1100

FIG. 11

	T12	T21	T23	T32	T31	T13
900	YES	YES	YES	YES	YES	YES
901	YES	YES	YES	YES	YES	NO
902	YES	YES	YES	YES	NO	YES
903	YES	YES	YES	NO	YES	YES
904	YES	YES	NO	YES	YES	YES
905	YES	NO	YES	YES	YES	YES
906	NO	YES	YES	YES	YES	YES
907	YES	YES	YES	NO	YES	NO
908	YES	YES	NO	YES	NO	YES
909	YES	NO	YES	YES	YES	NO
910	NO	YES	YES	YES	NO	YES
911	YES	NO	YES	NO	YES	YES
912	NO	YES	NO	YES	YES	YES
913	YES	YES	YES	YES	NO	NO
914	YES	YES	NO	NO	YES	YES
915	NO	NO	YES	YES	YES	YES
916	YES	NO	YES	NO	YES	NO
917	NO	YES	NO	YES	NO	YES

FIG. 9

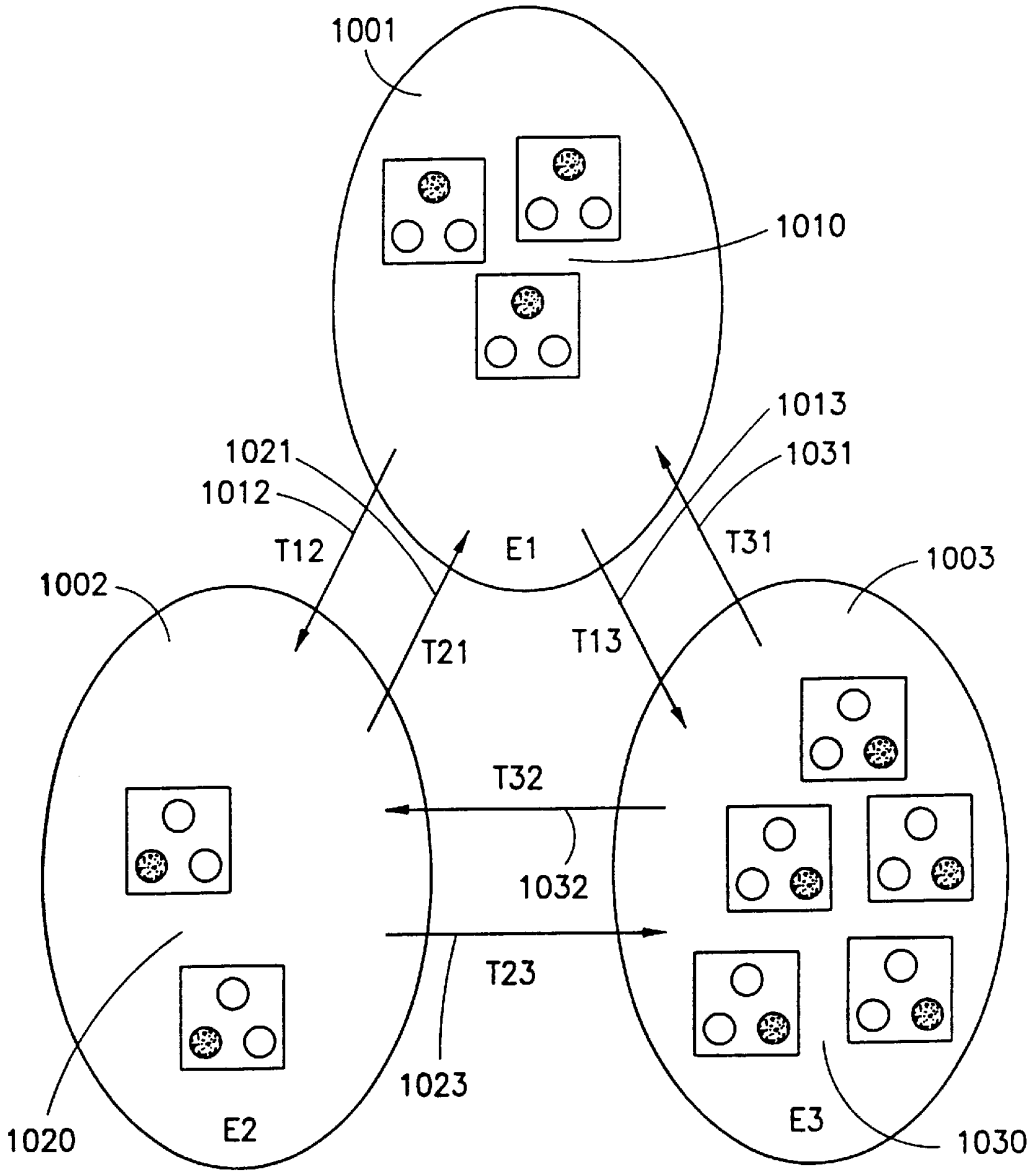

1001
1010
1013
1031
1021
1012
T12
E1
T31
1003
1002
T21
T13
T32
1032
T23
1023
1020
E2
E3
1030

FIG. 10

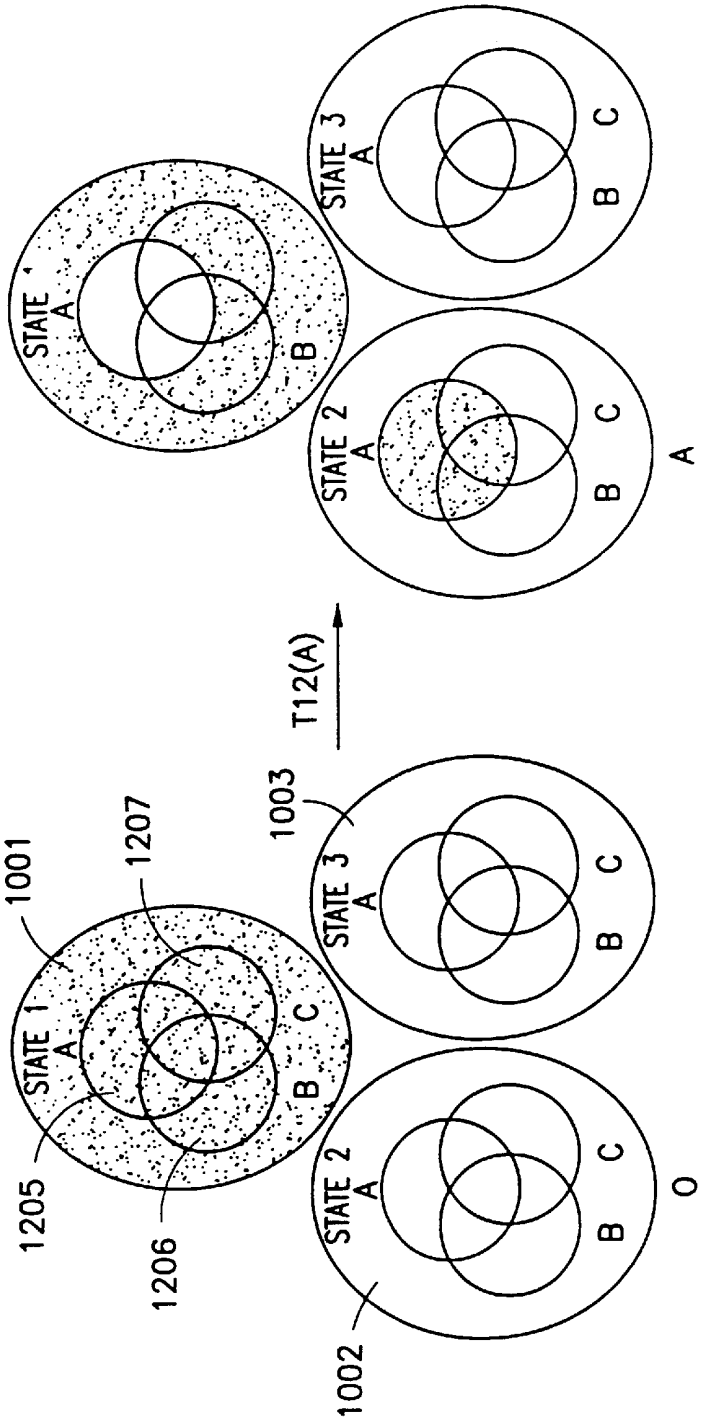

STATE 1
A
B
C
1001
1207
1205
1206
T12(A)
STATE 2
A
B
C
1002
STATE 3
A
B
C
1003
STATE 2
A
B
C
STATE 3
A
B
C
STATE 1
A
B

FIG. 12

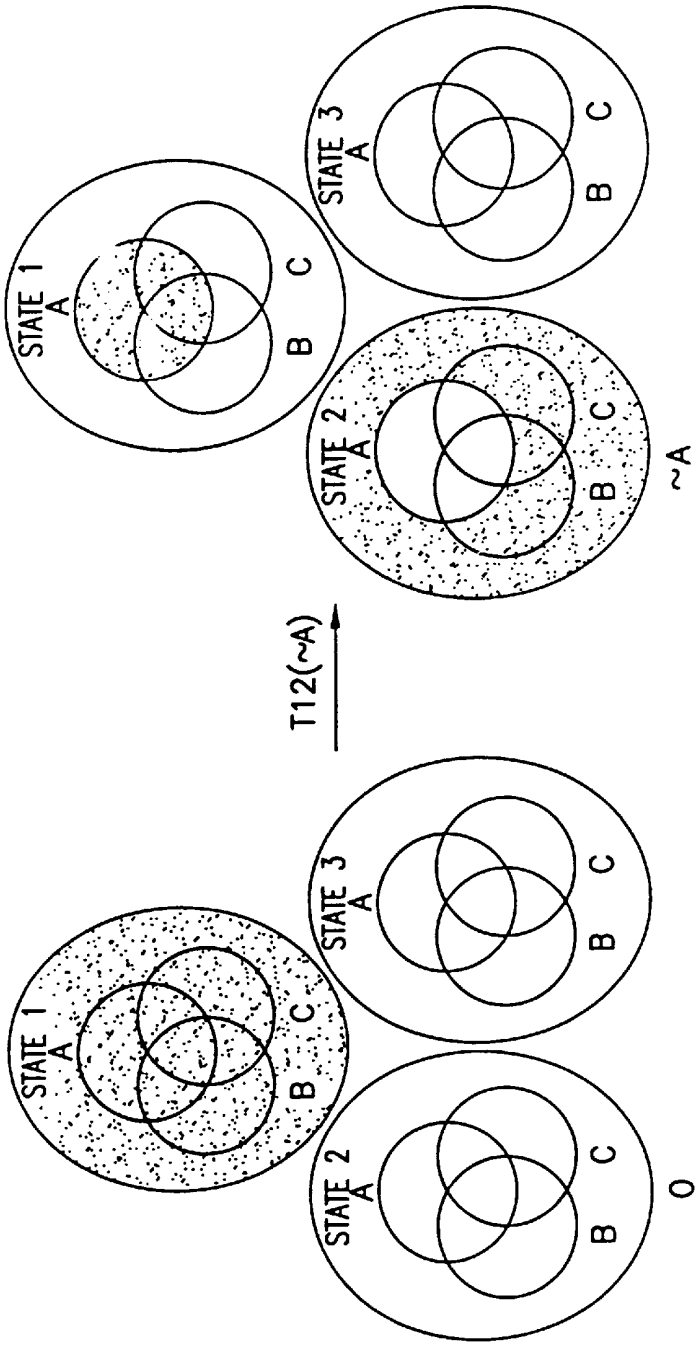

STATE 1
A
B
C
STATE 3
A
B
C
STATE 2
A
B
C
~A
T12(~A)
STATE 1
A
B
C
STATE 3
A
B
C
STATE 2
A
B
C
0

FIG. 13

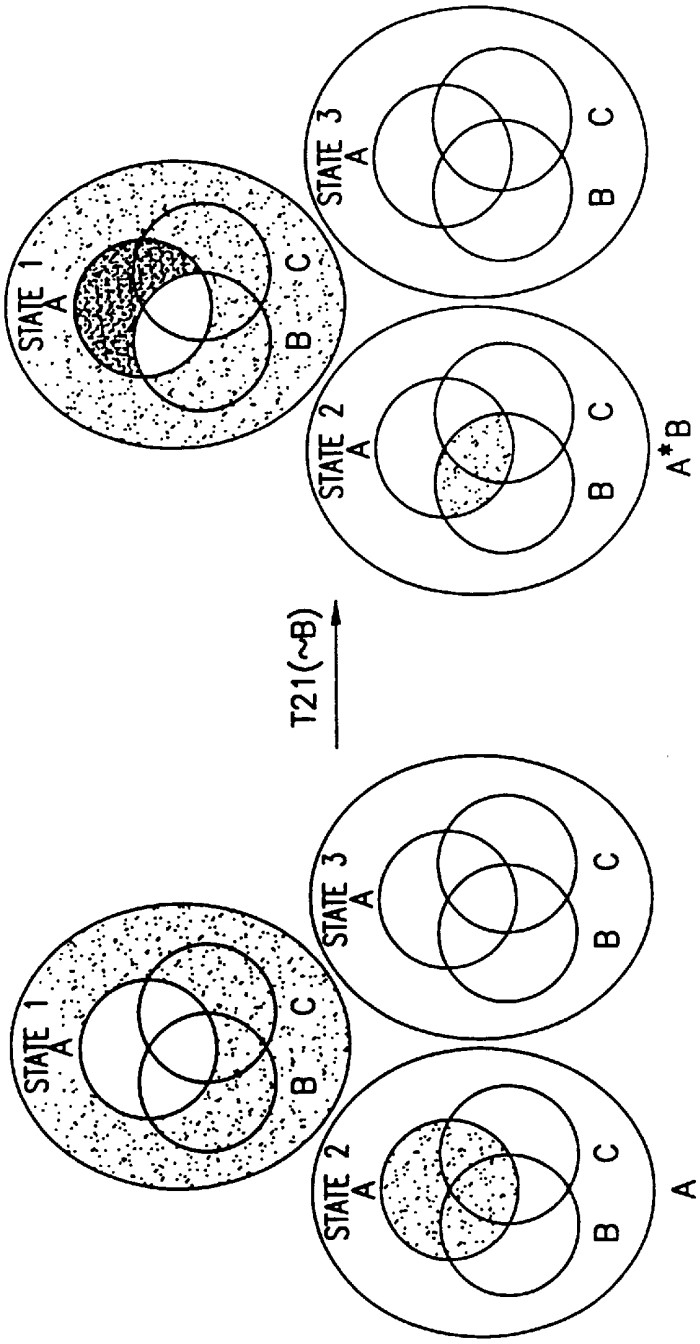

STATE 1
A
B
C
STATE 3
A
B
C
STATE 2
A
B
C
A*B
T21(~B)
STATE 1
A
B
C
STATE 3
A
B
C
STATE 2
A
B
C
A

FIG. 14

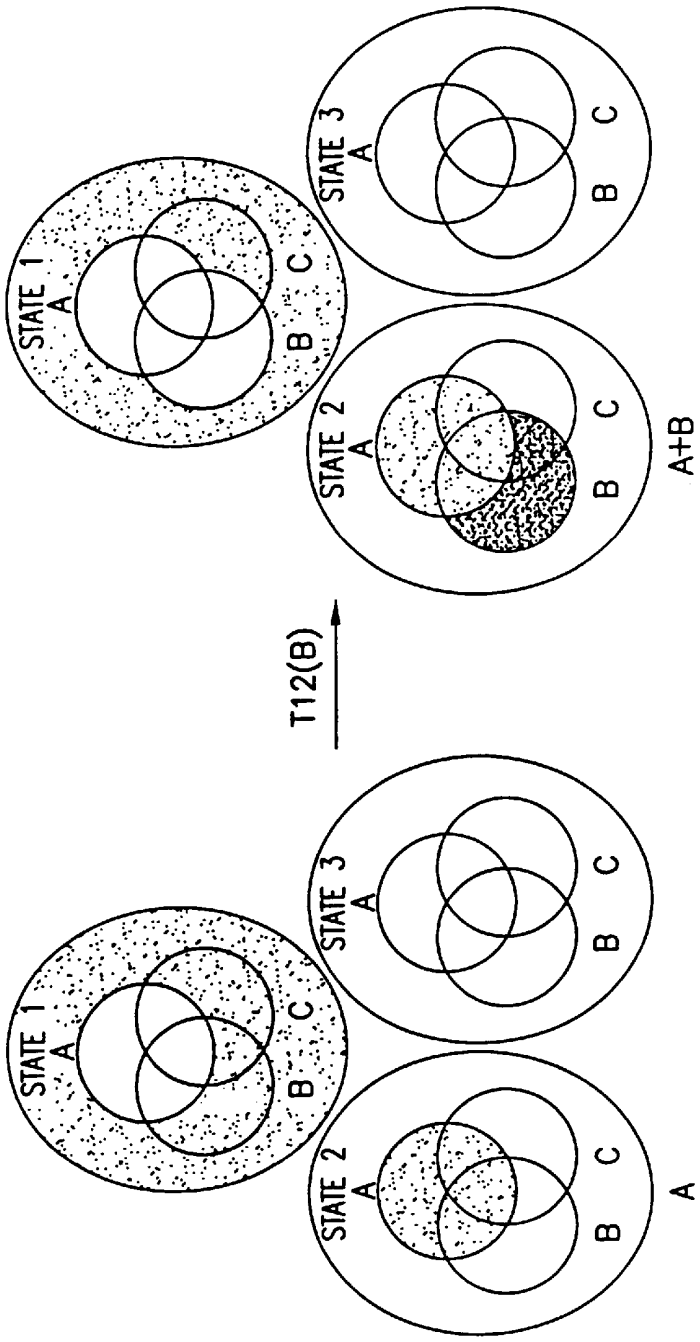

STATE 1
A
B
C
STATE 3
A
B
C
STATE 2
A
B
C
A+B
T12(B)
STATE 1
A
B
C
STATE 3
A
B
C
STATE 2
A
B
C
A

FIG. 15

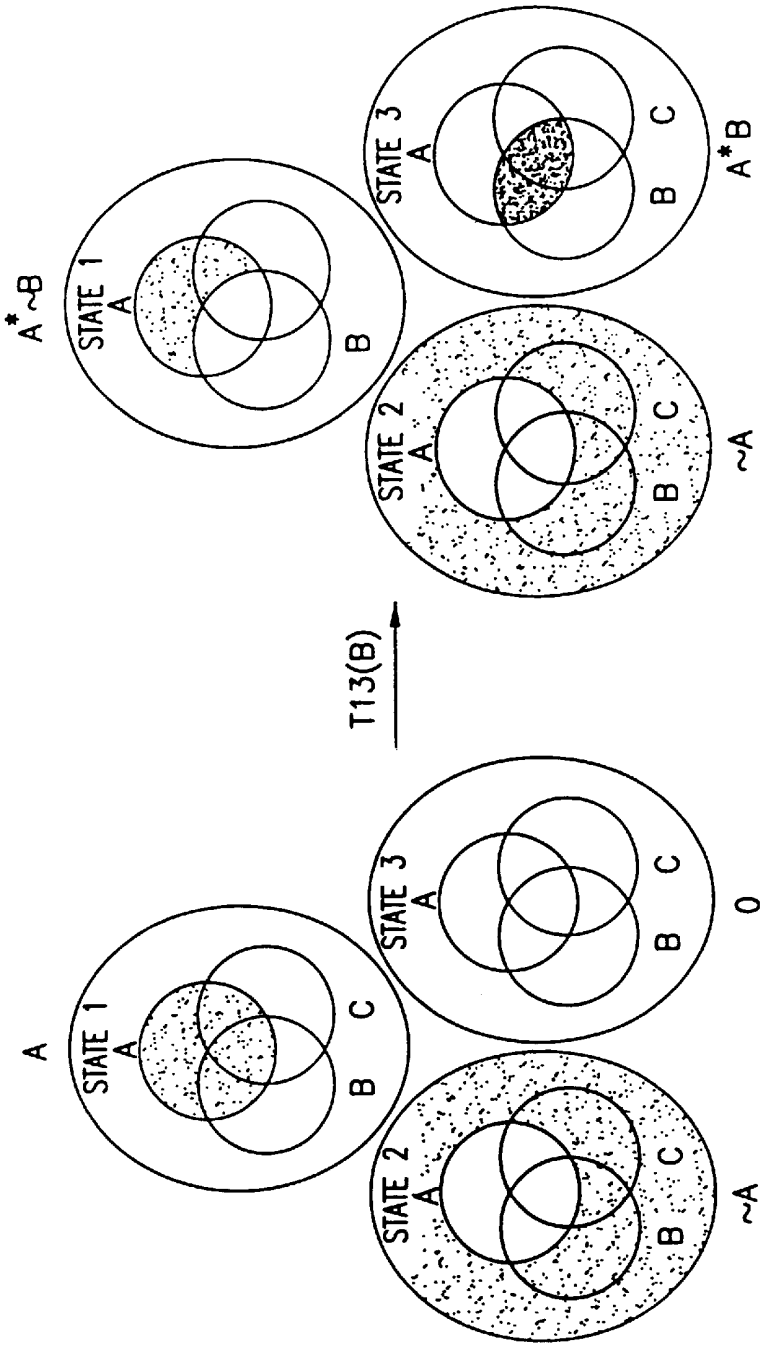

A*~B
STATE 1
A
B
STATE 3
A
B C
A*B
STATE 2
A
B C
~A
T13(B)
A
STATE 1
A
B C
STATE 3
A
B C
0
STATE 2
A
B C
~A

FIG. 16

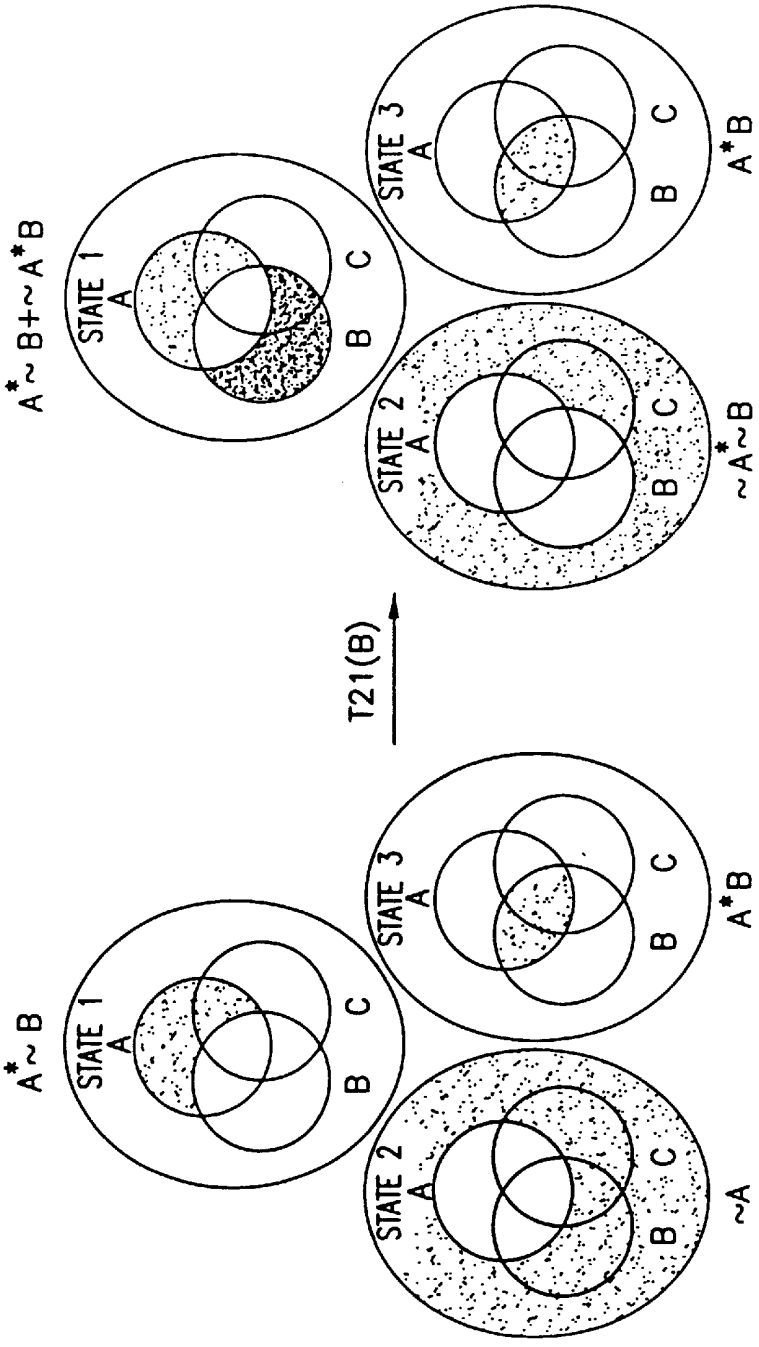

A*~B+~A*B
STATE 1
A
B C
STATE 3
A
B C
A*B
STATE 2
A
B C
~A*~B
T21(B)
A*~B
STATE 1
A
B C
STATE 3
A
B C
A*B
STATE 2
A
B C
~A

FIG. 17

	1810	STATE 1	STATE 2	STATE 3
1800		1	0	0
1801	T12(~A)	A	~A	0
1802	T13(B)	$A^*{\sim}B$	~A	A^*B
1803	T21(B)	$A^*{\sim}B+{\sim}A^*B$	$\sim A^*{\sim}B$	A^*B

FIG. 18

	1910	STATE 1	STATE 2	STATE 3
1900		1	0	0
1901	T12(A)	~A	A	0
1902	T23(B)	~A	$A^*{\sim}B$	A^*B
1903	T12(B)	$\sim A^*{\sim}B$	$A^*{\sim}B+{\sim}A^*B$	A^*B

FIG. 19

	2010	STATE 1	STATE 2	STATE 3
2000		1	0	0
2001	T13(A)	~A	0	A
2002	T32(B)	~A	A^*B	$A^*{\sim}B$
2003	T13(B)	$\sim A^*{\sim}B$	A^*B	$A^*{\sim}B+{\sim}A^*B$

FIG. 20

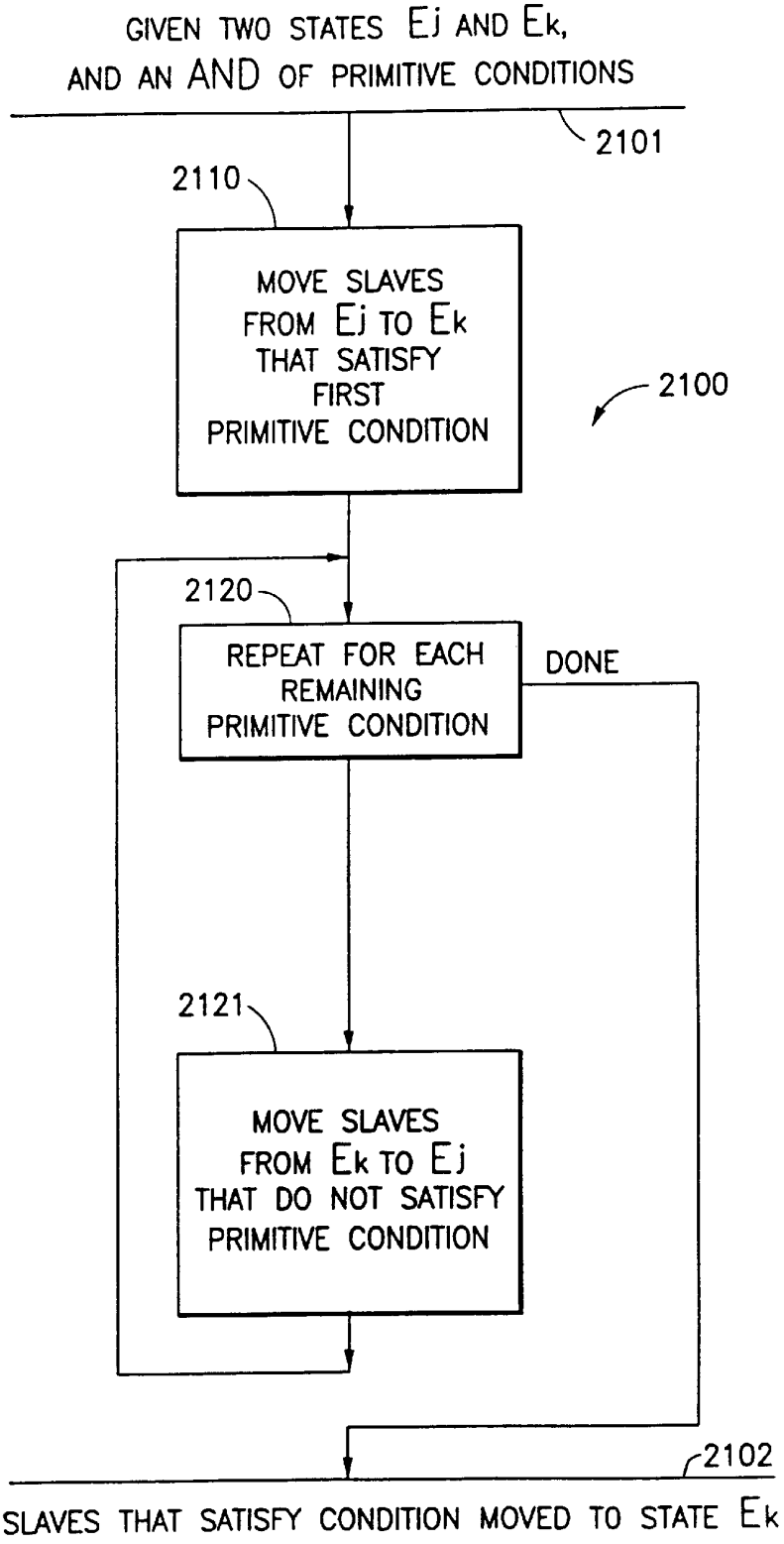

GIVEN TWO STATES Ej AND Ek,
AND AN AND OF PRIMITIVE CONDITIONS
2101
2110
MOVE SLAVES FROM Ej TO Ek THAT SATISFY FIRST PRIMITIVE CONDITION
2100
2120
REPEAT FOR EACH REMAINING PRIMITIVE CONDITION
DONE
2121
MOVE SLAVES FROM Ek TO Ej THAT DO NOT SATISFY PRIMITIVE CONDITION
2102
SLAVES THAT SATISFY CONDITION MOVED TO STATE Ek

FIG. 21

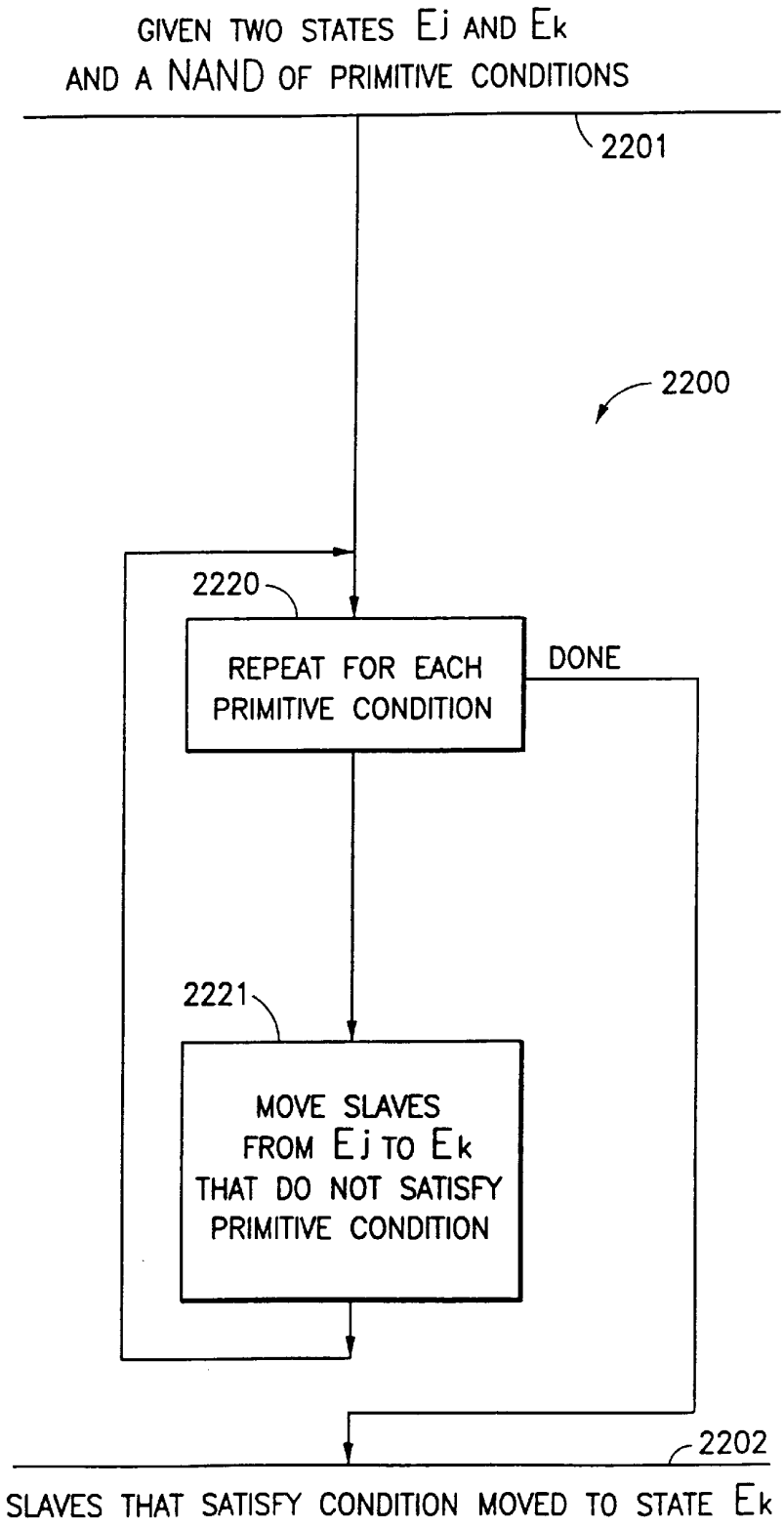

GIVEN TWO STATES Ej AND Ek
AND A NAND OF PRIMITIVE CONDITIONS
2201
2200
2220
REPEAT FOR EACH PRIMITIVE CONDITION
DONE
2221
MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY PRIMITIVE CONDITION
2202
SLAVES THAT SATISFY CONDITION MOVED TO STATE Ek

FIG. 22

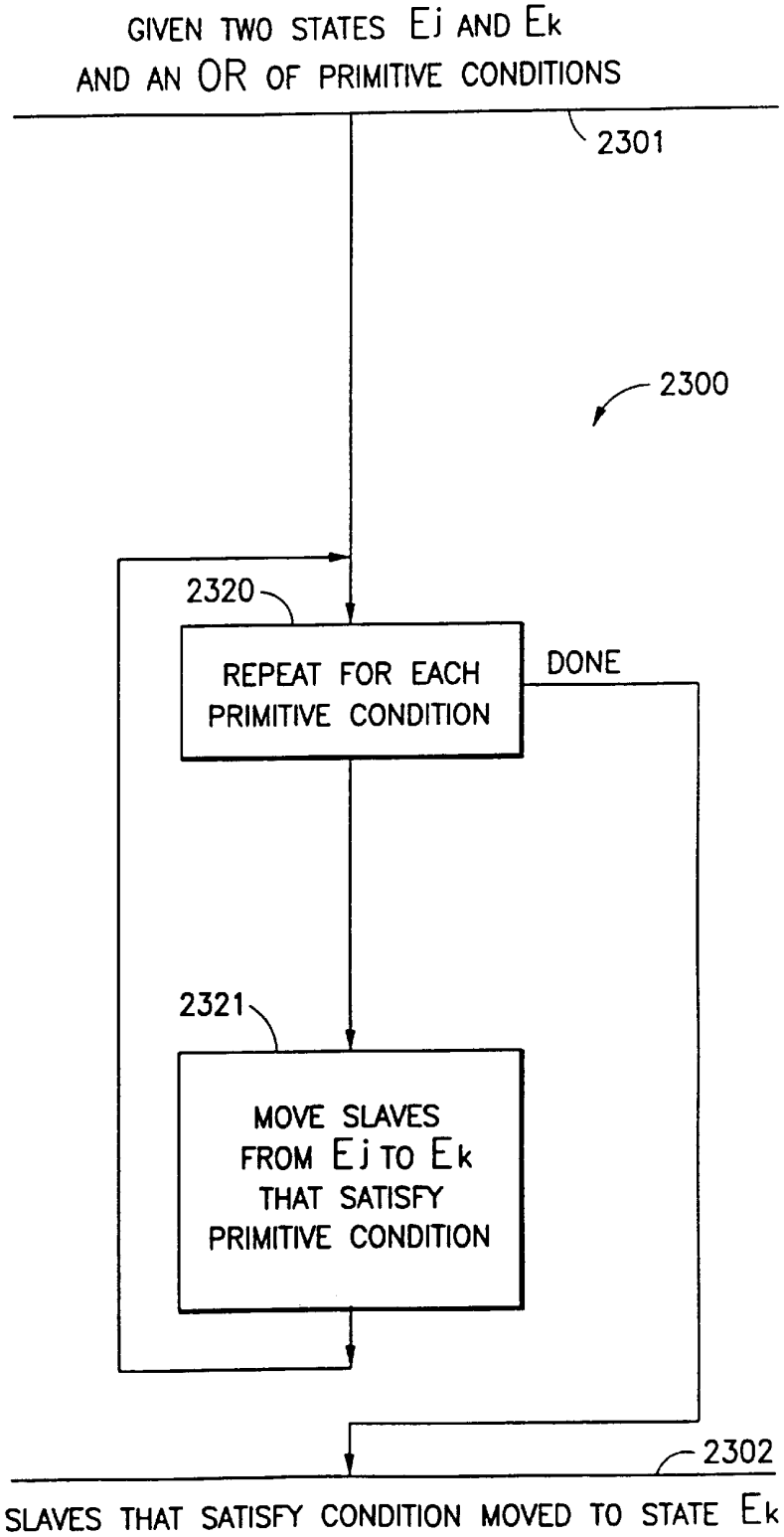

GIVEN TWO STATES Ej AND Ek
AND AN OR OF PRIMITIVE CONDITIONS
2301
2300
2320
REPEAT FOR EACH PRIMITIVE CONDITION
DONE
2321
MOVE SLAVES FROM Ej TO Ek THAT SATISFY PRIMITIVE CONDITION
2302
SLAVES THAT SATISFY CONDITION MOVED TO STATE Ek

FIG. 23

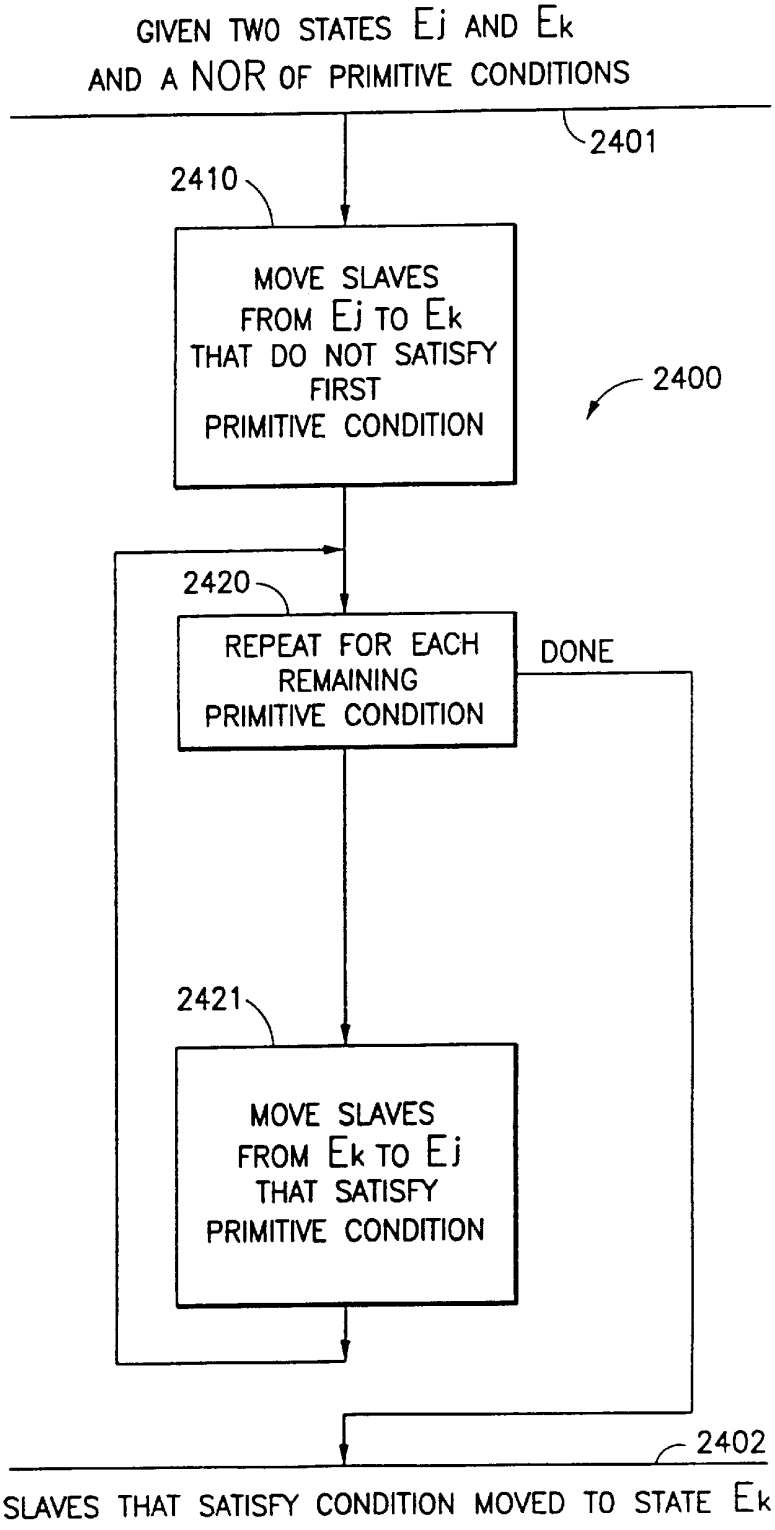

GIVEN TWO STATES Ej AND Ek
AND A NOR OF PRIMITIVE CONDITIONS
2401
2410
MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY FIRST PRIMITIVE CONDITION
2400
2420
REPEAT FOR EACH REMAINING PRIMITIVE CONDITION
DONE
2421
MOVE SLAVES FROM Ek TO Ej THAT SATISFY PRIMITIVE CONDITION
2402
SLAVES THAT SATISFY CONDITION MOVED TO STATE Ek

FIG. 24

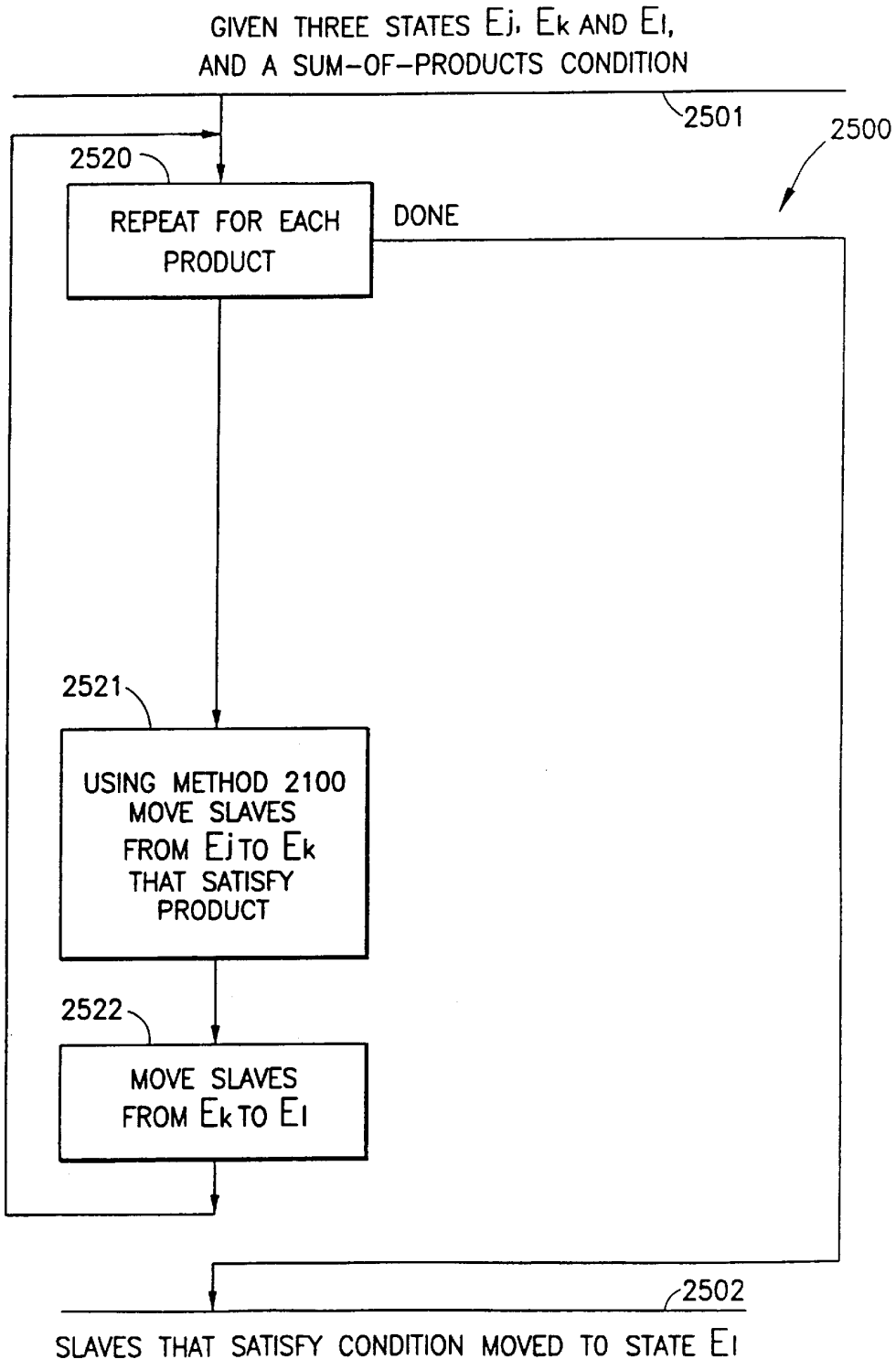

GIVEN THREE STATES Ej, Ek AND El,
AND A SUM-OF-PRODUCTS CONDITION
2501
2500
2520
REPEAT FOR EACH PRODUCT
DONE
2521
USING METHOD 2100 MOVE SLAVES FROM Ej TO Ek THAT SATISFY PRODUCT
2522
MOVE SLAVES FROM Ek TO El
2502
SLAVES THAT SATISFY CONDITION MOVED TO STATE El

FIG. 25

	2610	STATE 1	STATE 2	STATE 3
2600		1	0	0
2601	T12(~A)	A	~A	0
2602	T21(~B)	A+~B	~A*B	0
2603	T23(1)	A+~B	0	~A*B
2604	T12(A)	~A*~B	A	~A*B
2605	T21(B)	~A*~B+A*B	A*~B	~A*B
2606	T23(1)	~A*~B+A*B	0	~A*B+A*~B

FIG. 26

	2710	STATE 1	STATE 2	STATE 3
2700		1	0	0
2701	T12(A)	~A	A	0
2702	T21(~C)	~A+~C	A*C	0
2703	T23(1)	~A+~C	0	A*C
2704	T12(B)	~A*~B+~B*~C	~A*B+B*~C	A*C
2705	T21(~C)	~A*~B+~C	~A*B*C	A*C
2706	T23(1)	~A*~B+~C	0	A*C+B*C

FIG. 27

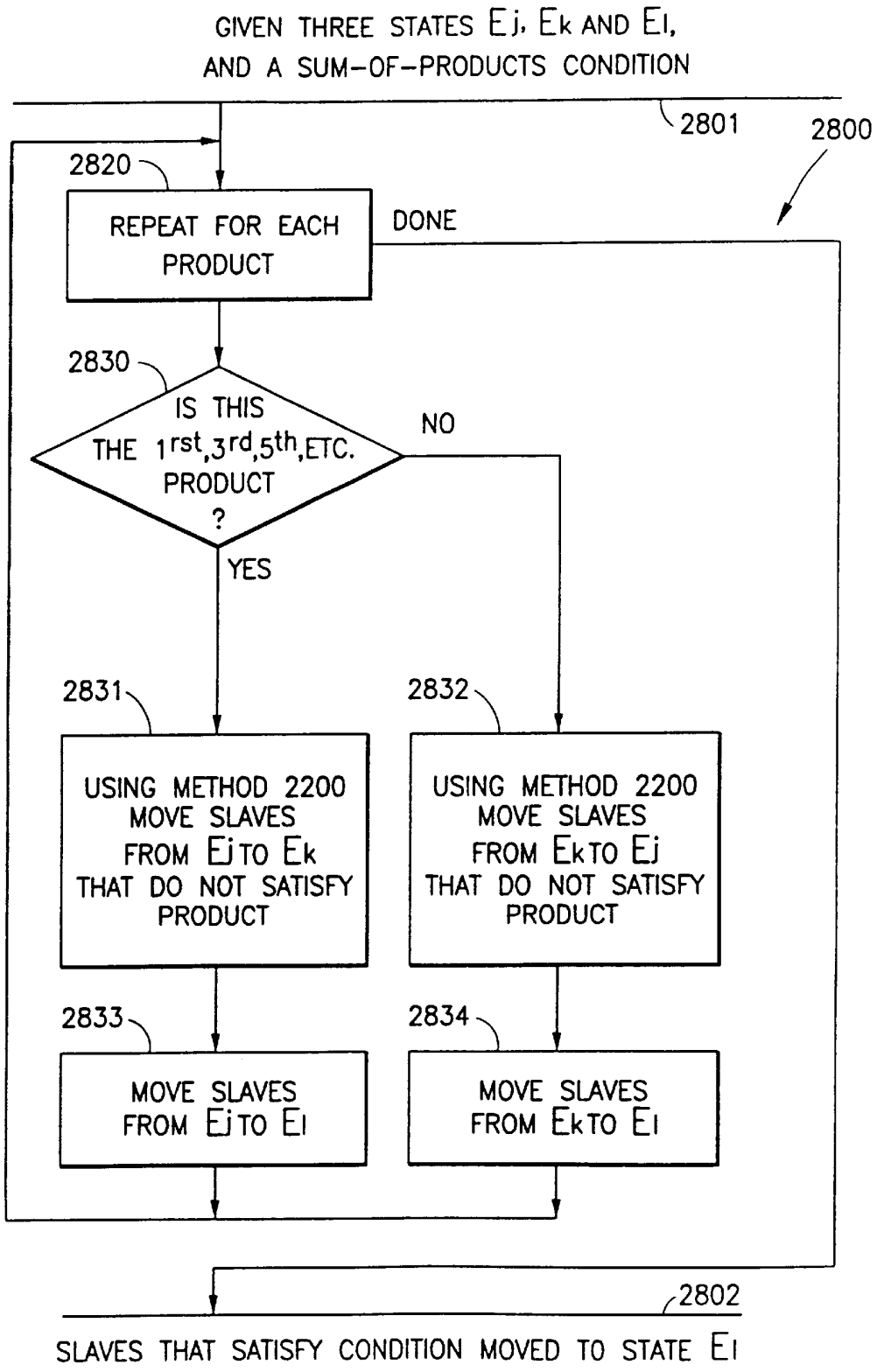

GIVEN THREE STATES Ej, Ek AND Ei,
AND A SUM-OF-PRODUCTS CONDITION
2801
2800
2820
REPEAT FOR EACH PRODUCT
DONE
2830
IS THIS THE 1rst,3rd,5th,ETC. PRODUCT ?
NO
YES
2831
USING METHOD 2200 MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY PRODUCT
2832
USING METHOD 2200 MOVE SLAVES FROM Ek TO Ej THAT DO NOT SATISFY PRODUCT
2833
MOVE SLAVES FROM Ej TO Ei
2834
MOVE SLAVES FROM Ek TO Ei
2802
SLAVES THAT SATISFY CONDITION MOVED TO STATE Ei

FIG. 28

	2910	STATE 1	STATE 2	STATE 3
2900		1	0	0
2901	T12(A)	~A	A	0
2902	T12(~B)	~A*B	A+~B	0
2903	T13(1)	0	A+~B	~A*B
2904	T21(~A)	~A*~B	A	~A*B
2905	T21(B)	~A*~B+A*B	A*~B	~A*B
2906	T23(1)	~A*~B+A*B	0	~A*B+A*~B

FIG. 29

	3010	STATE 1	STATE 2	STATE 3
3000		1	0	0
3001	T12(~A)	A	~A	0
3002	T12(~C)	A*C	~A+~C	0
3003	T23(1)	0	~A+~C	A*C
3004	T21(~B)	~A*~B+~B*~C	~A*B+B*~C	A*C
3005	T21(~C)	~A*~B+~C	~A*B*C	A*C
3006	T23(1)	~A*~B+~C	0	A*C+B*C

FIG. 30

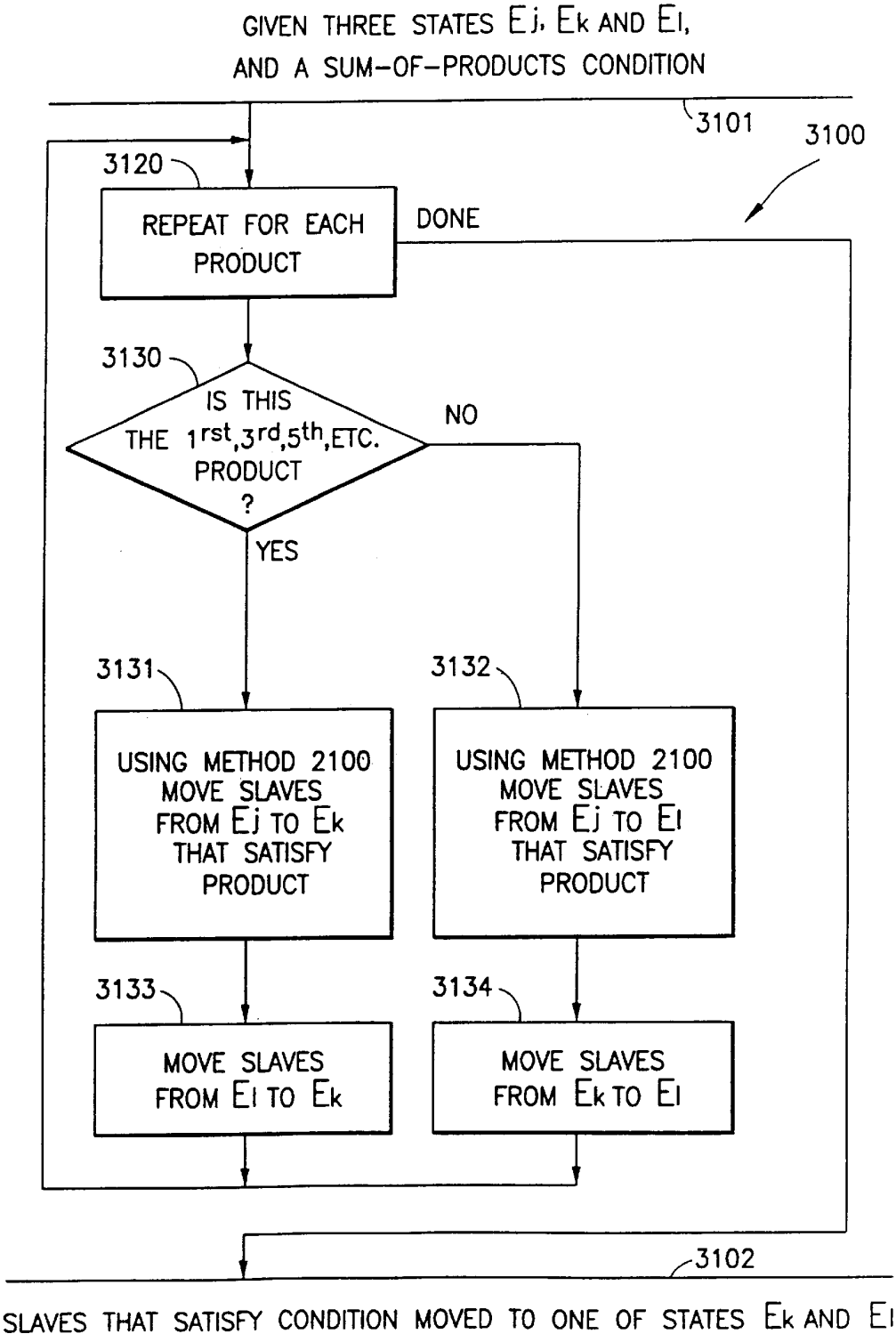

GIVEN THREE STATES Ej, Ek AND El,
AND A SUM-OF-PRODUCTS CONDITION
3101
3100
3120
REPEAT FOR EACH PRODUCT
DONE
3130
IS THIS THE 1rst,3rd,5th,ETC. PRODUCT ?
NO
YES
3131
USING METHOD 2100 MOVE SLAVES FROM Ej TO Ek THAT SATISFY PRODUCT
3132
USING METHOD 2100 MOVE SLAVES FROM Ej TO El THAT SATISFY PRODUCT
3133
MOVE SLAVES FROM El TO Ek
3134
MOVE SLAVES FROM Ek TO El
3102
SLAVES THAT SATISFY CONDITION MOVED TO ONE OF STATES Ek AND El

FIG. 31

	3210	STATE 1	STATE 2	STATE 3
3200		1	0	0
3201	T12(~A)	A	~A	0
3202	T21(~B)	A+~B	~A*B	0
3203	T32(1)	A+~B	~A*B	0
3204	T13(A)	~A*~B	~A*B	A
3205	T31(B)	~A*~B+A*B	~A*B	A*~B
3206	T23(1)	~A*~B+A*B	0	~A*B+A*~B

FIG. 32

	3310	STATE 1	STATE 2	STATE 3
3300		1	0	0
3301	T12(A)	~A	A	0
3302	T21(~C)	~A+~C	A*C	0
3303	T32(1)	~A+~C	A*C	0
3304	T13(B)	~A*~B+~B*~C	A*C	~A*B+B*~C
3305	T31(~C)	~A*~B+~C	A*C	~A*B*C
3306	T23(1)	~A*~B+~C	0	A*C+B*C

FIG. 33

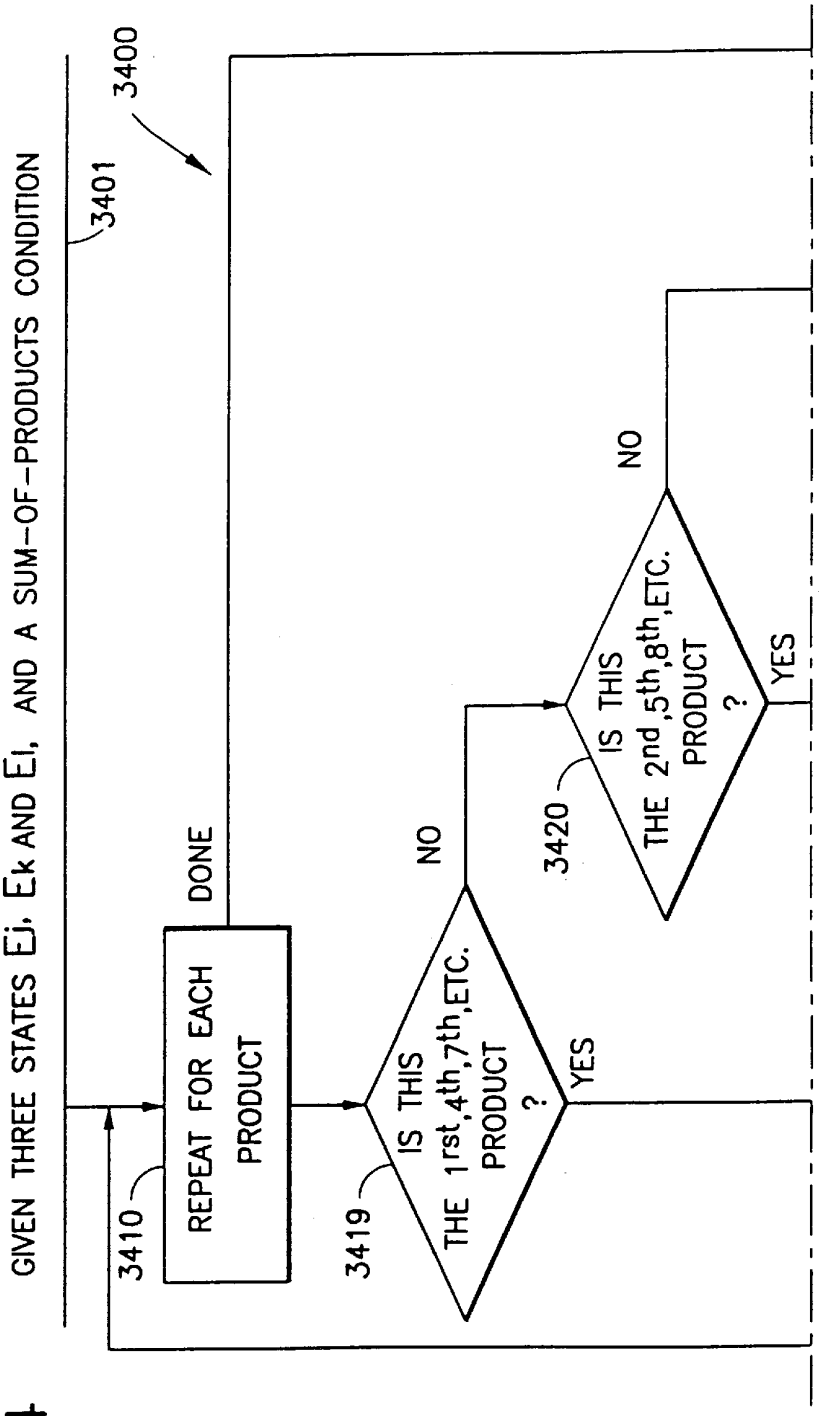


FIG. 34A

FIG. 34

FIG. 34A
FIG. 34B

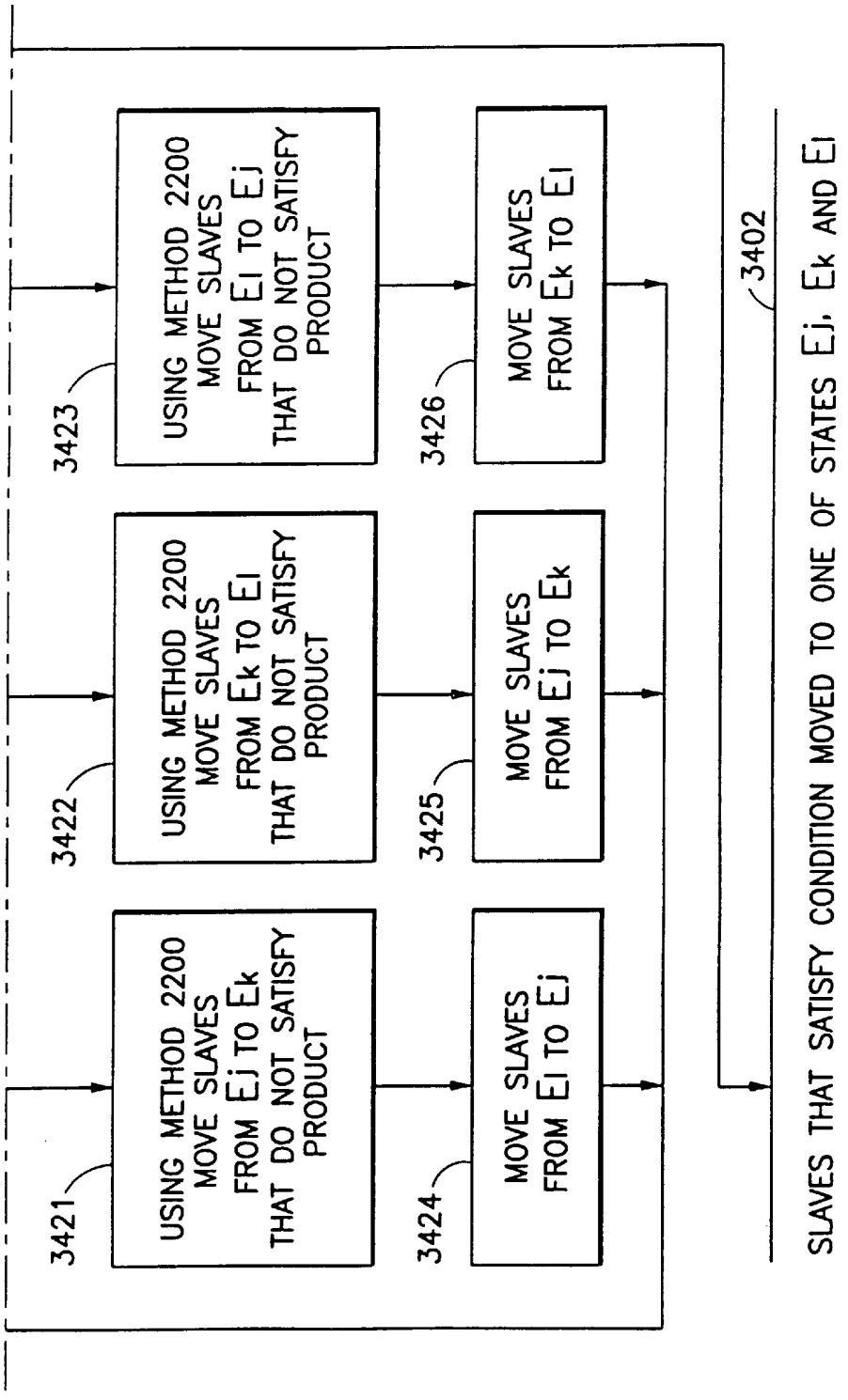

USING METHOD 2200 MOVE SLAVES FROM Ei TO Ej THAT DO NOT SATISFY PRODUCT
3423
MOVE SLAVES FROM Ek TO Ei
3426
USING METHOD 2200 MOVE SLAVES FROM Ek TO Ei THAT DO NOT SATISFY PRODUCT
3422
MOVE SLAVES FROM Ej TO Ek
3425
USING METHOD 2200 MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY PRODUCT
3421
MOVE SLAVES FROM Ei TO Ej
3424
3402
SLAVES THAT SATISFY CONDITION MOVED TO ONE OF STATES Ej, Ek AND Ei

FIG. 34B

	3510	STATE 1	STATE 2	STATE 3
3500		1	0	0
3501	T12(A)	~A	A	0
3502	T12(~B)	~A*B	A+~B	0
3503	T31(1)	~A*B	A+~B	0
3504	T23(~A)	~A*B	A	~A*~B
3505	T23(B)	~A*B	A*~B	~A*~B+A*B
3506	T12(1)	0	~A*B+A*~B	~A*~B+A*B

FIG. 35

	3610	STATE 1	STATE 2	STATE 3
3600		1	0	0
3601	T12(~A)	A	~A	0
3602	T12(~C)	A*C	~A+~C	0
3603	T31(1)	A*C	~A+~C	0
3604	T23(~B)	A*C	~A*B+B*~C	~A*~B+~B*~C
3605	T23(~C)	A*C	~A*B*C	~A*~B+~C
3606	T12(1)	0	A*C+B*C	~A*~B+~C

FIG. 36

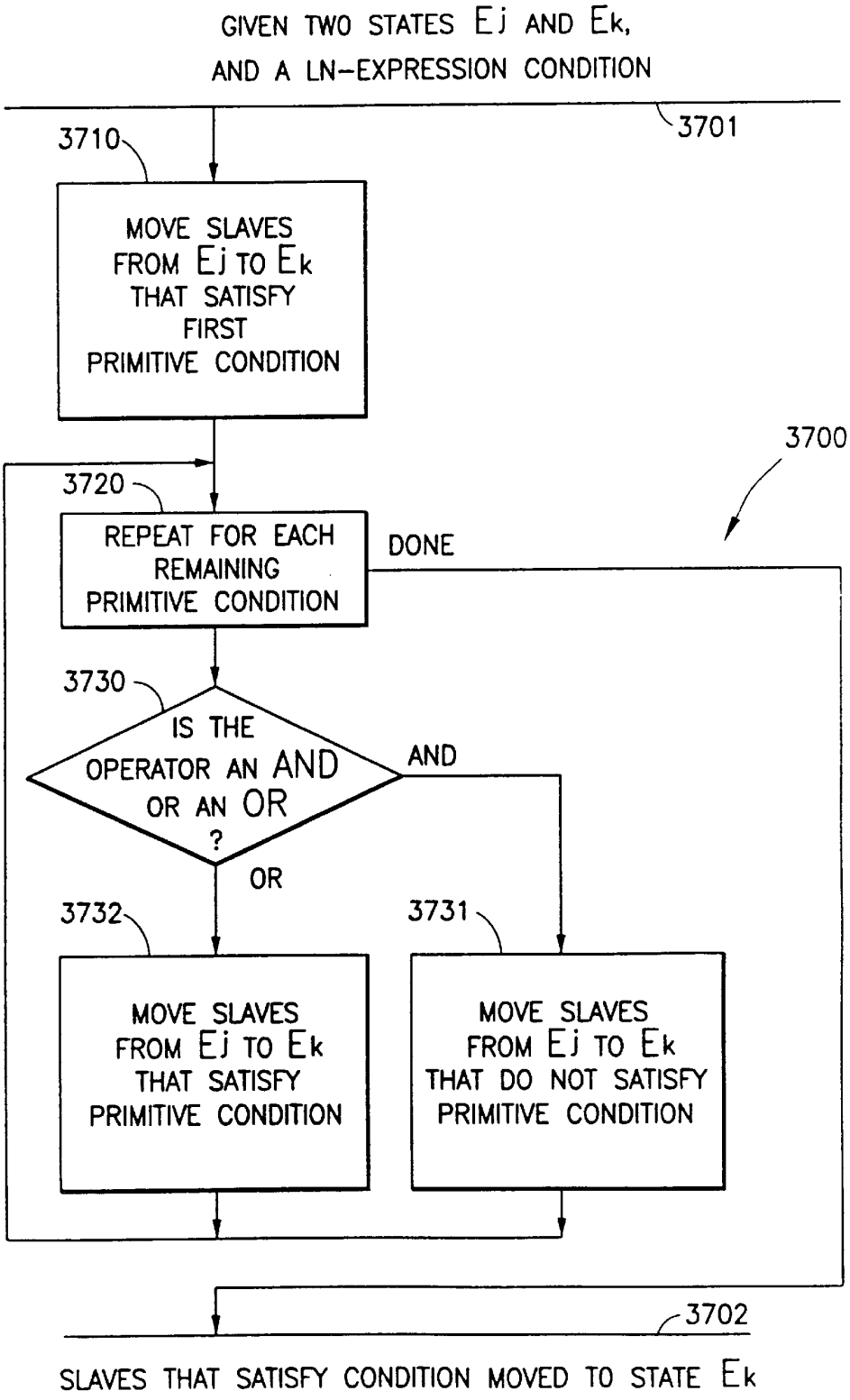

GIVEN TWO STATES Ej AND Ek,
AND A LN-EXPRESSION CONDITION
3701
3710
MOVE SLAVES FROM Ej TO Ek THAT SATISFY FIRST PRIMITIVE CONDITION
3700
3720
REPEAT FOR EACH REMAINING PRIMITIVE CONDITION
DONE
3730
IS THE OPERATOR AN AND OR AN OR ?
AND
OR
3732
MOVE SLAVES FROM Ej TO Ek THAT SATISFY PRIMITIVE CONDITION
3731
MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY PRIMITIVE CONDITION
3702
SLAVES THAT SATISFY CONDITION MOVED TO STATE Ek

FIG. 37

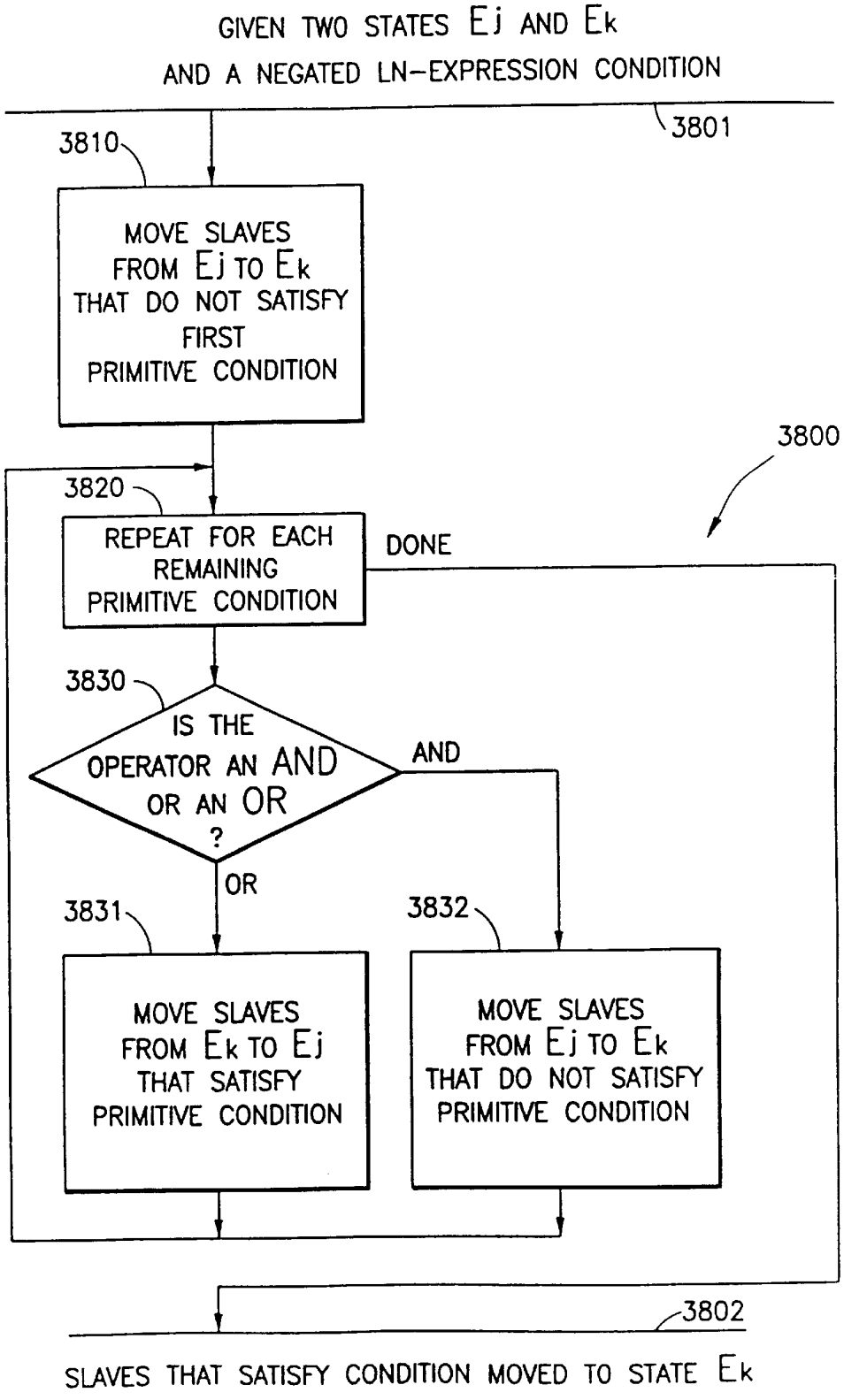

GIVEN TWO STATES Ej AND Ek
AND A NEGATED LN-EXPRESSION CONDITION
3801
3810
MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY FIRST PRIMITIVE CONDITION
3800
3820
REPEAT FOR EACH REMAINING PRIMITIVE CONDITION
DONE
3830
IS THE OPERATOR AN AND OR AN OR ?
AND
OR
3831
MOVE SLAVES FROM Ek TO Ej THAT SATISFY PRIMITIVE CONDITION
3832
MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY PRIMITIVE CONDITION
3802
SLAVES THAT SATISFY CONDITION MOVED TO STATE Ek

FIG. 38

3910	E1	E2	E3
3900	1	0	0
3901 T12(A)	~A	A	0
3902 T12(B)	~A*~B	A+B	0
3903 T21(~C)	~A*~B+~C	(A+B)*C	0

FIG. 39

4010	E1	E2	E3
4000	1	0	0
4001 T12(~A)	A	~A	0
4002 T21(B)	A+B	~A*~B	0
4003 T12(~C)	(A+B)*C	~A*~B+~C	0

FIG. 40

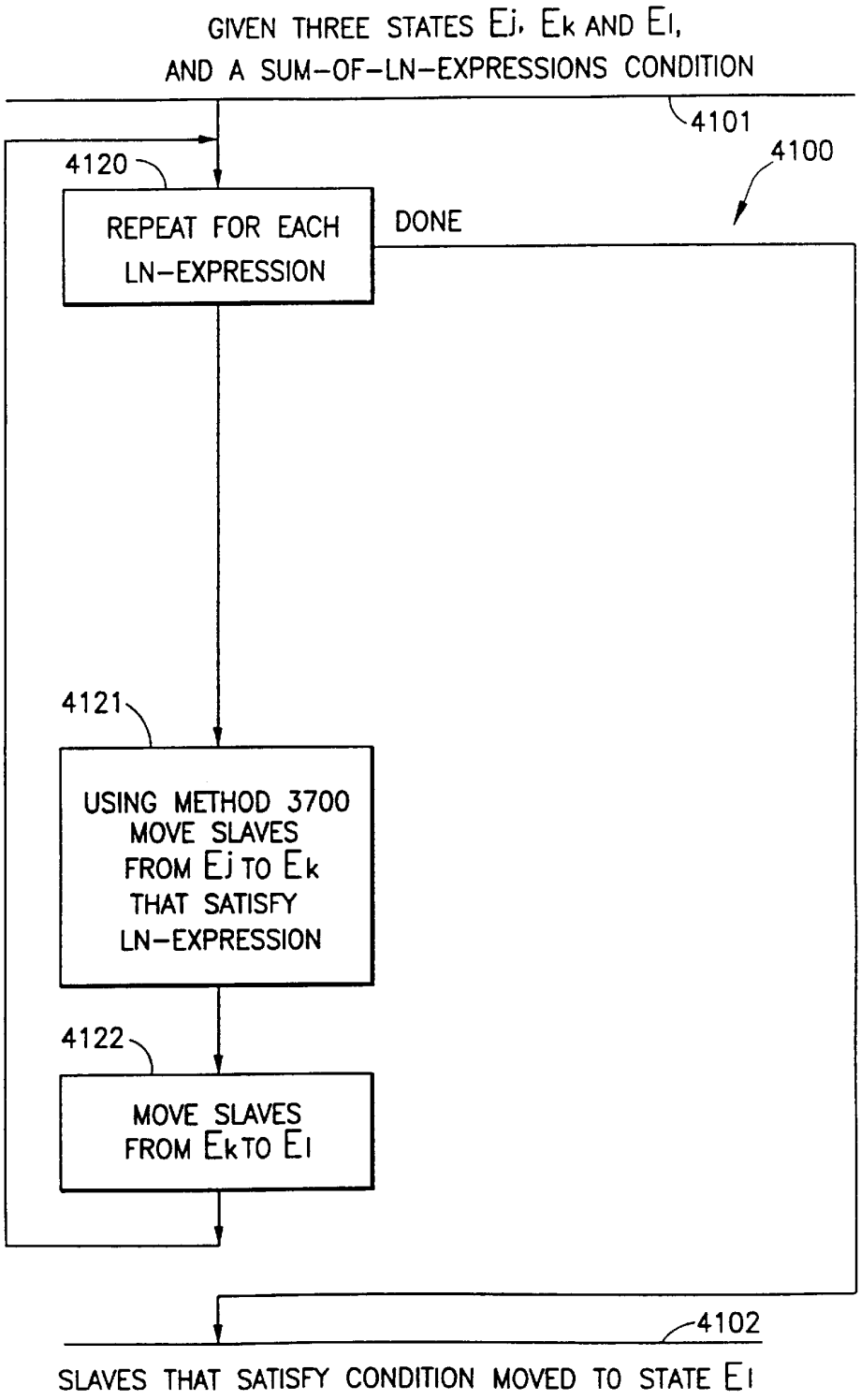

GIVEN THREE STATES Ej, Ek AND El,
AND A SUM-OF-LN-EXPRESSIONS CONDITION
4101
4100
4120
REPEAT FOR EACH LN-EXPRESSION
DONE
4121
USING METHOD 3700 MOVE SLAVES FROM Ej TO Ek THAT SATISFY LN-EXPRESSION
4122
MOVE SLAVES FROM Ek TO El
4102
SLAVES THAT SATISFY CONDITION MOVED TO STATE El

FIG. 41

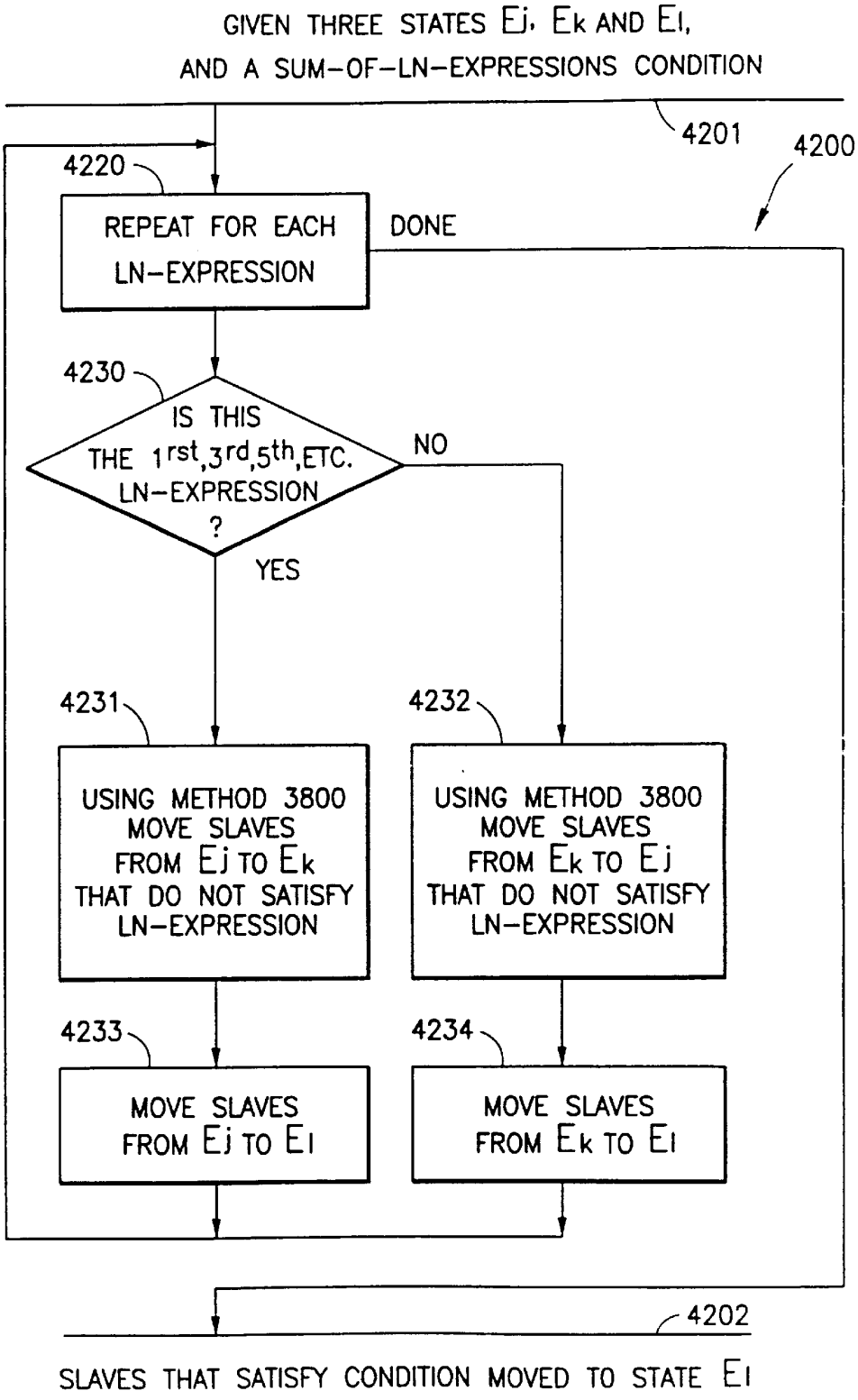

GIVEN THREE STATES Ej, Ek AND El,
AND A SUM-OF-LN-EXPRESSIONS CONDITION
4201
4200
4220
REPEAT FOR EACH LN-EXPRESSION
DONE
4230
IS THIS THE 1rst,3rd,5th,ETC. LN-EXPRESSION ?
NO
YES
4231
USING METHOD 3800 MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY LN-EXPRESSION
4232
USING METHOD 3800 MOVE SLAVES FROM Ek TO Ej THAT DO NOT SATISFY LN-EXPRESSION
4233
MOVE SLAVES FROM Ej TO El
4234
MOVE SLAVES FROM Ek TO El
4202
SLAVES THAT SATISFY CONDITION MOVED TO STATE El

FIG. 42

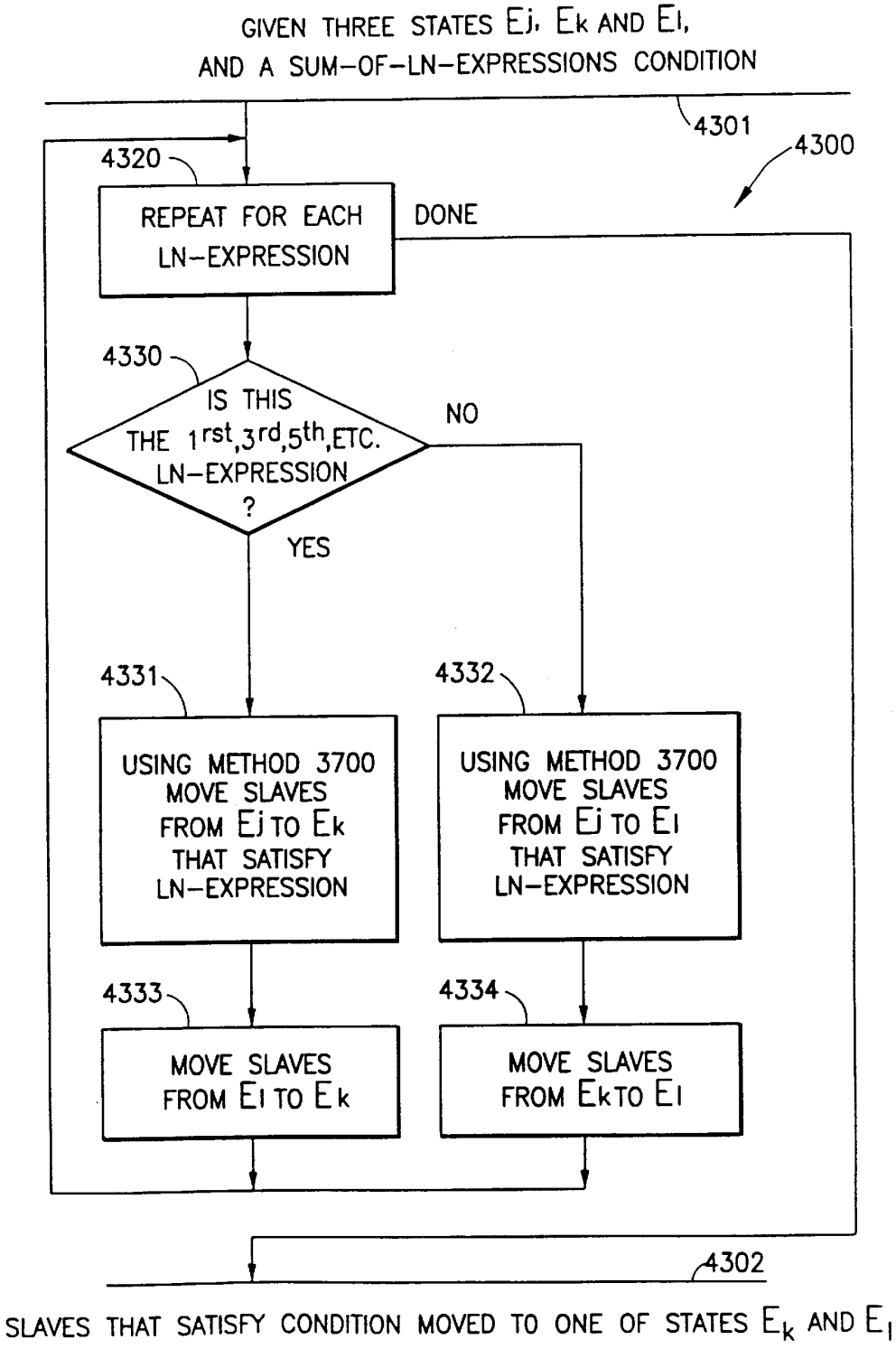

GIVEN THREE STATES Ej, Ek AND El,
AND A SUM-OF-LN-EXPRESSIONS CONDITION
4301
4300
4320
REPEAT FOR EACH LN-EXPRESSION
DONE
4330
IS THIS THE 1rst,3rd,5th,ETC. LN-EXPRESSION ?
NO
YES
4331
USING METHOD 3700 MOVE SLAVES FROM Ej TO Ek THAT SATISFY LN-EXPRESSION
4332
USING METHOD 3700 MOVE SLAVES FROM Ej TO El THAT SATISFY LN-EXPRESSION
4333
MOVE SLAVES FROM El TO Ek
4334
MOVE SLAVES FROM Ek TO El
4302
SLAVES THAT SATISFY CONDITION MOVED TO ONE OF STATES Ek AND El

FIG. 43

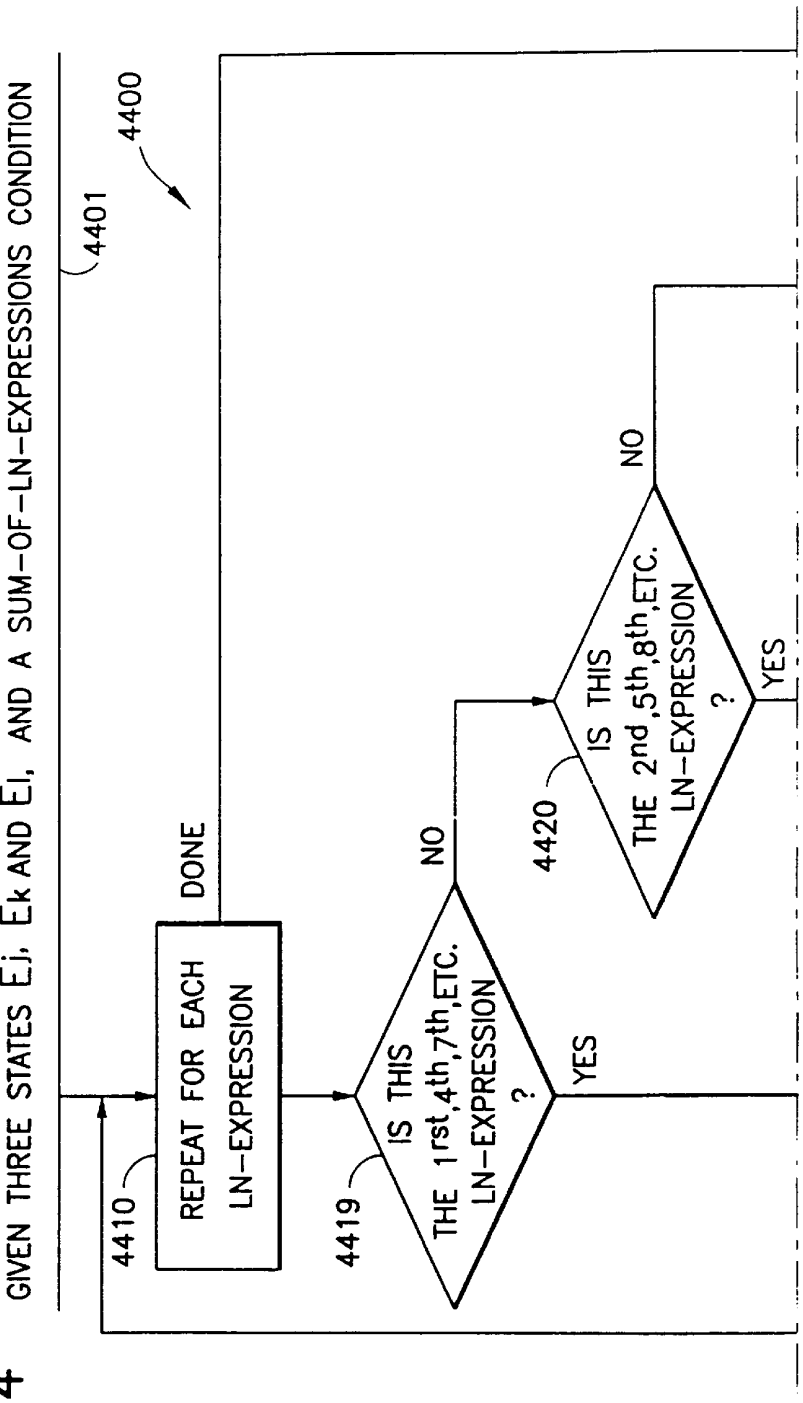


FIG. 44A

FIG. 44

FIG. 44A
FIG. 44B

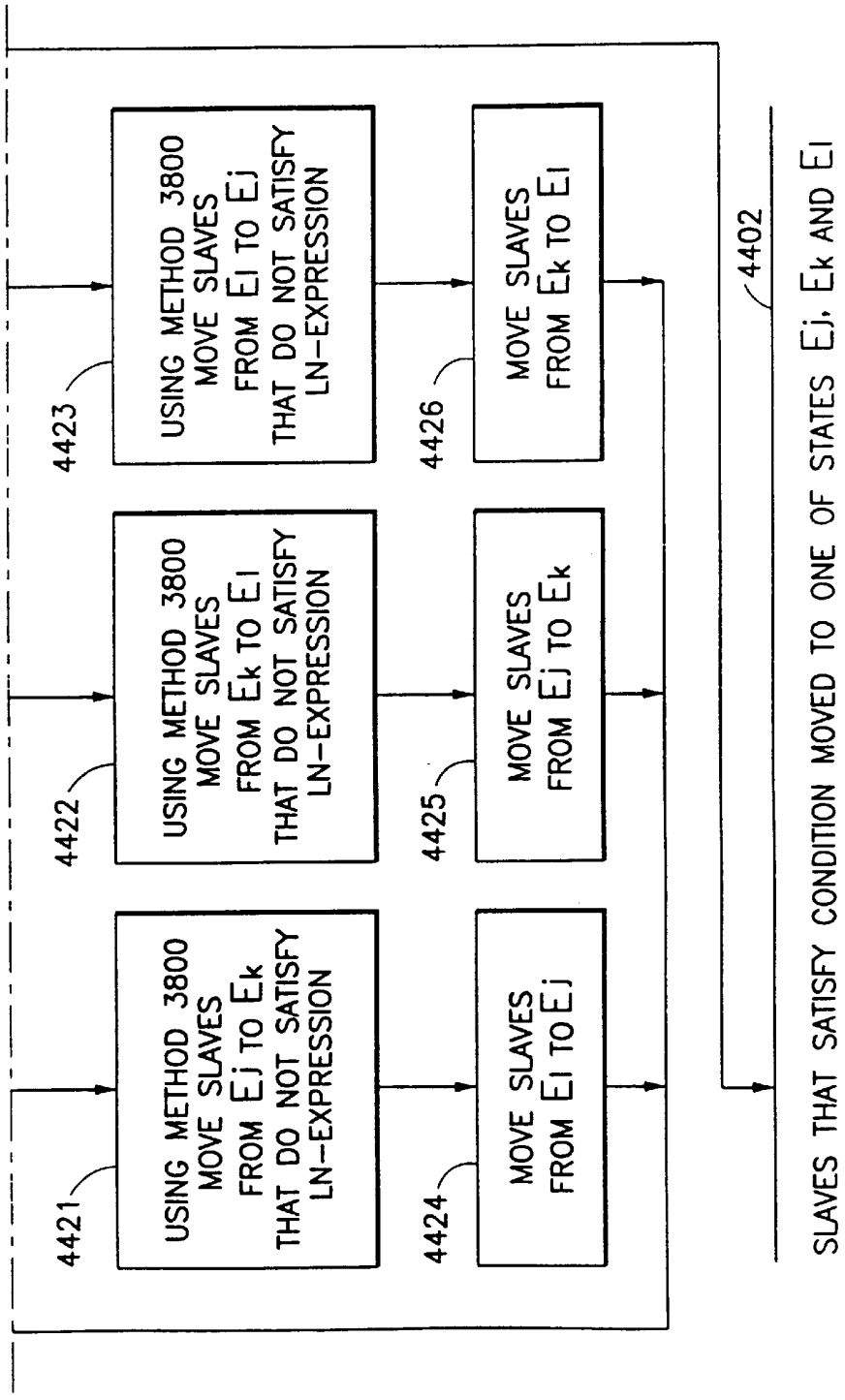

4421
USING METHOD 3800 MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY LN-EXPRESSION
4422
USING METHOD 3800 MOVE SLAVES FROM Ek TO Ei THAT DO NOT SATISFY LN-EXPRESSION
4423
USING METHOD 3800 MOVE SLAVES FROM Ei TO Ej THAT DO NOT SATISFY LN-EXPRESSION
4424
MOVE SLAVES FROM Ei TO Ej
4425
MOVE SLAVES FROM Ej TO Ek
4426
MOVE SLAVES FROM Ek TO Ei
4402
SLAVES THAT SATISFY CONDITION MOVED TO ONE OF STATES Ej, Ek AND Ei

FIG. 44B

SYSTEM AND METHOD FOR SELECTING A SUBSET OF AUTONOMOUS AND INDEPENDENT SLAVE ENTITIES

FIELD OF THE INVENTION

The invention relates to communications between a master station and one or more slave stations. More specifically, the invention relates to a master station selecting subset(s) of the slave stations by broadcasting commands with conditions that the selected slaves meet.

BACKGROUND OF THE INVENTION

In the prior art, a master control unit is to communicate with a plurality of autonomous and independent slaves. In such environment, the number of slaves is often not known a priori. There may in fact be no slaves with which the master can communicate. Among the reasons the master may have to communicate with the slaves are (a) the need to acknowledge their presence, (b) identify and count them and/or (c) order them to perform tasks. This kind of computational environment falls under the broader category of broadcasting sequential processes, which is defined by Narain Gehani in Chapter 9 of the book co-edited with Andrew McGettrick, "Concurrent Programming" (Addison-Wesley, 1988), which is herein incorporated by reference in its entirety.

Because the master often does not know ahead of time the number of slaves present and because that number may be very large and possibly unyieldy, it is advantageous for the master to be able to select a subset of the slaves with whom to communicate further. Such a selection must of course be done by a conditional. Those slaves that meet the condition are thus considered selected, while those that do not meet the condition are considered not selected. The selection is performed by broadcasting to all slaves the condition that must be met. This is akin to asking those among a large crowd of people whose last name is Lowell to raise their hand. Each slave is defined has having at least the capability to listen to the master's broadcasts, to receive the broadcast condition and to self-test so as to determine whether it meets the condition. See for example, U.S. patent application No. 08/303,965, entitled "Radio Frequency (RF) Group Select Protocol" to Cesar et al. filed on Sep. 9, 1994 which is herein incorporated by reference in its entirety.

Practical environments where this computational model can be applied include bus arbitration, wireless communication, distributed and parallel processing.

Characteristic of such environments is the existence of a protocol for how master and slaves communicate. The aforementioned capability of subset selection can be an important additional component of that protocol.

Finite state-machines are a well-known modelling tool. The set theory that often accompanies the definition of finite state-machines is also well known. Both subjects are amply covered in any of many books on discrete or finite mathematics that are available today. The book by Ralph Grimaldi, "Discrete and Combinatorial Mathematics: An Applied Introduction" (Addison-Wesley, 1985), is a fine example of its kind.

STATEMENT OF PROBLEMS WITH THE PRIOR ART

In the prior art, the methods used for selecting subsets of slaves is limited to comparisons against the information held by the slaves such that the comparison is either true or false and the slaves can be in either of two selection states: selected or not selected. The slave may contain many other states, but only two are effectively dedicated to the purpose of subset selection.

In some older prior art, a comparison will override a previous comparison. There is no notion of accumulating and combining successive comparisons to effect a selection according to a complex condition.

More recent prior art allows slaves to move between two selection states according to successive comparisons. That allows some complex conditions to be effected. However not all complex conditions can be effected with such two-selection-state machine. For example, the complex condition "is-red and is-not-tall or is-not-red and is-tall", that is, the EXCLUSIVE-OR of the two simple comparisons "is-red" and "is-tall", can not be performed such that the subset of slaves that satisfy the EXCLUSIVE-OR are in the first state and those that do not satisfy the EXCLUSIVE-OR are in the second state. In the case of complex conditions involving two comparisons and their negation, the two-selection-state machine can not perform the EXCLUSIVE-OR and the EQUIVALENCE logical operators. In the case of complex conditions involving more than two comparisons and their negation, the two-selection-state machine can not perform an increasingly large number of logical equations.

In the prior art, conditions such as the EXCLUSIVE-OR must be broken up into two independent processing steps. First, slaves satisfying the first AND term are selected and all necessary processing sequence is performed over them. Second, after a general reset, slaves satisfying the second AND term are selected and the same necessary processing sequence is repeated over those. That means that the processing sequence must be broadcast twice. In the case of more complicated conditions, rebroadcasting of such sequence may happen more than twice. For example, the condition (A*~B*~C)+(~A*B*~C)+(~A*~B+C) would need three rebroadcasts.

The only conditions that can be executed in a single round of broadcasting by a two-selection-state logic as used by the prior art are those conditions that can be expressed by a left-nested expression, such as ((A+B+C)*D*E)+F). OR conditions, such as (A+B+C+D), and AND conditions, such as (A*B*C), are particular cases of left-nested expressions. In contrast, EXCLUSIVE-OR type conditions, such as (A*B*~C)+(A*~B*C)*(~A*B*C), can not be written as left-nested expressions and therefore can not be handled by the two-selection-state logic.

Among the prior art is U.S. Pat. No. 5,434,572, entitled "System and Method for Initiating Communications between a Controller and a Selected Subset of Multiple Transponders in a Common RF Field" to Smith dated Jul. 18, 1995. The method begins by selecting all "transponder" in a field and, by a sequence of commands, incrementally moving groups of slaves to a "reset" condition.

In the U.S. Pat. No. 5,410,315, entitled "Group-Addressable Transponder Arrangement" to Huber dated Apr. 25, 1995, a selection is essentially made through a comparison against a "group and/or unit address". Unlike Smith, there is no progressive incremental refinement. Huber can perform only limited AND operations.

OBJECTS OF THE INVENTION

An object of this invention is a system and method for using arbitrarily complex logical conditions to select slave stations that satisfy those conditions transmitted by a master station through a series one or more commands.

An object of this invention is a system and method for using arbitrarily complex logical conditions to select RF transponders that satisfy those conditions transmitted by a base station through a series one or more commands.

SUMMARY OF THE INVENTION

The present invention is a system and method for selecting a subset of a plurality of autonomous and independent slaves, wherein each slave comprises (i) a three-state machine dedicated to selection, (ii) some other stored information, and (iii) a logic to execute externally provided commands in a command sequence that exercise the three-state machine. The primary purpose of the commands is to effect state transitions. The slave receives the command, which causes a comparison to be performed against the slave's stored information, the results of which possibly causing a state transition in the slave.

The commands, in a sequence called a command sequence, are broadcast from at least one master control unit to zero or more slaves. The exact number of slaves may not be known by the master. The master executes a method by which a sequence of discrete commands is broadcast to all slaves. The overall purpose of the method is to bring a subset of the slaves to be at the same state of their three-state machine, while all other slaves are at any one of the two other remaining states.

A three-state machine dedicated to selection is present in every slave. Each slave is at one of those three states, therefore, at any one time, the slaves can be sub-divided into three subsets: those slaves that have their selection three-state machine at the first state, those at the second state, and those at the third state. In a preferred embodiment, transitions are possible between any two states of the three-state machine. Transitions are requested by command (sequence) broadcast from the master. A command specifies a desired transition, say from the second state to the first state. Only slaves that are at the second state may be affected. The command also specifies a condition under which the transition will occur. If the condition is met, the transition is effected; if not, the slave remains in its previous state.

In a preferred embodiment, slaves can be moved from a first state to a second state and visa versa. Only slaves in the second state can be moved to a third state. The slaves in the third state ignore the remaining commands in the command sequence. In alternative preferred embodiments, the first and second states reverse roles after an end of one or more subsequences in the sequences of commands. Also, the second and third states can reverse roles after an end of one or more subsequences. Further, the states of the slaves can cycle their roles at the end of one or more of the subsequences.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of preferred embodiments of the invention with reference to the drawings that are included:

FIG. **1** is a block diagram of a master control unit broadcasting commands to a plurality of slaves.

FIG. **2** is a block diagram of the components of a master control unit.

FIG. **3** is a block diagram of the components of a slave.

FIG. **4** shows a state diagram showing a three-state machine that allows all six possible transitions between any two different states.

FIG. **5** shows a state diagram showing a three-state machine that allows five possible transitions between any two different states.

FIG. **6** shows a state diagram showing a three-state machine that allows four possible transitions between any two different states, such that any state is reachable from any other state.

FIG. **7** shows a state diagram showing a three-state machine that allows four possible transitions between any two different states, such that any state is reachable from any other state and two of the states have no transitions between them.

FIG. **8** shows a state diagram showing a three-state machine that allows three possible transitions between any two different states, such that any state is reachable from any other state.

FIG. **9** lists all possible three-state machines that can be used for the purposes of this invention.

FIG. **10** is a set theoretic representation of how the plurality of slaves is subdivided into at most three sets.

FIG. **11** describes what happens when a command to transfer slaves that satisfy some condition from one state to another is broadcast.

FIGS. **12**, **13**, **14**, **15**, **16** and **17** exemplifies by means of Venn diagrams how a command transfers elements from one set to another.

FIGS. **18**, **19** and **20** show sequences of commands for selecting a subset of slaves that satisfy an EXCLUSIVE-OR condition and such that those slaves end up in a first, second and third set, respectively

FIG. **21** describes a method that computes a product (AND) condition.

FIG. **22** describes a method that computes a negated product (NAND) condition.

FIG. **23** describes a method that computes a sum (OR) condition.

FIG. **24** describes a method that computes a negated sum (NOR) condition.

FIG. **25** describes a method that computes a condition written in sum-of-products form whereby the product terms are built in a second state and the sum is accumulated in a third state.

FIGS. **26** and **27** exemplify the use of the method described in FIG. **25**.

FIG. **28** describes a method that computes a condition written in sum-of-products form whereby the product terms are built alternatively in a first and second states and the sum is accumulated in a third state.

FIGS. **29** and **30** exemplify the use of the method described in FIG. **28**.

FIG. **31** describes a method that computes a condition written in sum-of-products form whereby the product terms are built alternatively in a second and third states and the sum is accumulated in that second and third state, respectively.

FIGS. **32** and **33** exemplify the use of the method described in FIG. **31**.

FIG. **34** describes a method that computes a condition written in sum-of-products form whereby the product terms are built alternatively in a first, second and third states and the sum is accumulated in that first, second and third state, respectively.

FIGS. **35** and **36** exemplify the use of the method described in FIG. **31**.

FIG. **37** describes a method that computes a single left-nested expression using only two states.

FIG. **38** describes a method that computes the negation of a single left-nested expression using only two states.

FIG. **39** exemplifies the use of the method described in FIG. **37**.

FIG. **40** exemplifies the use of the method described in FIG. **38**.

FIG. **41** describes a method that computes a condition written in sum-of-left-nested-expressions form whereby the left-nested expressions are built in a second state and the sum is accumulated in a third state.

FIG. **42** describes a method that computes a condition written in sum-of-left-nested-expressions form whereby the left-nested expressions are built alternatively in a first and second states and the sum is accumulated in a third state.

FIG. **43** describes a method that computes a condition written in sum-of-left-nested-expressions form whereby the left-nested expressions are built alternatively in a second and third states and the sum is accumulated in that second and third states, respectively.

FIG. **44** describes a method that computes a condition written in sum-of-left-nested-expressions form whereby the left-nested expressions are built in a first, second and third states and the sum is accumulated in that first, second and third states, respectively.

DETAILED DESCRIPTION OF THE INVENTION

FIG. **1** shows least one master control unit **101** communicating with a plurality of autonomous and independent slaves **102**. The communication medium **103** can be in terms of either direct electric contact means or electromagnetic radiation means, e.g., radio frequency (RF) embodiments. However, it also encompasses light, sound and other frequencies utilized for signalling purposes. Master unit **101** is capable of broadcasting commands at the plurality of slaves **102** via communication medium **103**. Each slave is capable of receiving the broadcast commands which are processed by logic **150**. For example, U.S. Pat. No. 4,656,463 to Anders et al. shows an RF tag systems where for our purposes the active transceiver (AT) would be the master control unit **101**, the passive transceivers (PT) would be the independent slaves **102**, and the communication medium **103** would be RF over free space. In an alternative preferred embodiment, U.S. Pat. No. 5,371,852 to Attanasio et al. shows a cluster of computers where for our purposes the gateway would be the master unit **101**, the nodes in the cluster would be the slaves **102**, and the communication medium **103** would be the interconnect. These references are incorporated by reference in their entirety.

FIG. **2** shows that a master unit **200** comprises (a) means **201** for broadcasting commands from a command set **250** to a plurality of slaves, and (b) processing means **202** for determining the correct sequence of commands to be broadcast. In one embodiment, processing means **202** are proximate to broadcast means **201**. Other embodiments are possible where processing means **202** and broadcast means **201** are remote from each other.

FIG. **3** shows that a slave **300** comprises (a) a receiver **301** for receiving commands from a master, (b) a three-state machine **304**, (c) a memory with stored information **303** (d) processor or logic **302** for executing any received command, performing any condition testing over the stored information **303** as specified by the command, and effecting a state transition on three-state machine **304** conditional to the result of the condition testing. Receiving means **301** and broadcasting means **201** must be necessity be compatible. The receiving means **301** are well known. For example in RF tagging, radio frequency receivers/transmitters are used. In the networking arts standard connections to networks and/or information buses are used. Stored information **303** is typically embodied in the form of registers and/or memory.

The three-state machine **304** comprises a select state, a first unselect state, and a second unselect state state. All three of these states can be used in the selection and unselection of sets of slaves. The logic **302** is further described below.

The significant difference between a two and a three-selection-state machine is that, using any sequence of commands, the former can only isolate or select slaves that satisfy a condition expressed by a left-nested expression. Using a two-selection-state machine, there is no sequence of commands that can process conditions that are expressed as as a SUM-of-left-nested-expressions.

Only a three-selection-state machine can select slaves that satisfy a condition expressed by a sum-of-left-nested-expression. Further, a three-selection-state machine is also sufficient to select a set of slaves that satisfy any arbitrary condition, even though those conditions are expressed by a sum-of-left-nested-expression. Therefore adding a fourth, fifth, etc. state does not add any new capability to the selection logic. In addition, since any condition can be expressed by a sum-of-left-nested expression, a three-selection-state machine can select a set of slave satisfying any possible condition. This capability is undisclosed and unrecognized in the prior art.

The invention enables this capability because a separate condition (or set of conditions), each corresponding to a set of slaves, can be isolated in any one of the three states at any given time. Therefore, operations on two sets of conditions, in two of the respective states, can be performed without affecting or being affected by the conditions held in the third state.

In one embodiment, receiving means **301**, processing means **302**, stored information **303**, and three-state machine **304** are proximate. Other embodiments are possible where the components **301**, **302**, **303** and **304** are remote from each other, in part or completely.

The three states are dedicated to the process of determining whether a slave satisfies an arbitrarily complex condition. During the process of determining whether the slave satisfies the condition, the slave may be in any of the three states as dictated by the process. If the slave does not satisfy the condition, the process assures that the slave will end up at a state that enables the slave to communicate further with the master.

Three-state machine **304** is part of every slave. A preferred three-state machine is shown in FIG. **4**. Three-state machine **400** includes the three states **401**, **402** and **403**, and all six possible state-to-state transitions **412**, **413**, **421**, **423**, **431** and **432**, where the transitions are between two different states. The three states **401**, **402** and **403** are named S1, S2 and S3, respectively. A transition from a state to itself is not helpful for the methodology described herein.

Other three-state machines are possible. In FIG. **5**, three-state machine **500** has only five of the six state-to-state transitions present in three-state machine **400**. Three-state machine **500** represents a class of six possible three-state machines where one of the six possible state-to-state transitions is inoperative. In the particular case of FIG. **5**, transition **413** is inoperative.

In FIG. **6**, three-state machine **600** has only four of the six state-to-state transitions of three-state machine **400**. The four operative transitions are such that any state can be reached from another state by means of one or two transitions. Moreover there is at least one possible transition between any two states. Three-state machine **600** represents a class of six possible three-state machines were two of the six possible state-to-state transitions are inoperative, while still providing access to any state from any other state. In the particular case of FIG. **6**, transitions **413** and **432** are inoperative.

In FIG. **7**, three-state machine **700** has only four of the six state-to-state transitions of three-state machine **400**. The four operative transitions are such that any state can be reached from another state by means of one or two transitions. Moreover there are two states between which there is no possible transition. Three-state machine **700** represents a class of three possible three-state machines where one of the pairs of transitions between two states is inoperative. In the particular case of FIG. **7**, the pair of transitions **413** and **431** between states **401** and **403** is inoperative.

In FIG. **8**, three-state machine **800** has only three of the six state-to-state transitions of three-state machine **400**. The three operative transitions are such that any state can be reached from another state by means of one or two transitions. Therefore each pair of states is connected by one and only one transition, in such a way that all transitions move either clockwise or anticlockwise. Three-state machine **800** represents a class of two possible three-state machines that have only three of the six state-to-state transitions. In the particular case of FIG. **8**, the three operative transitions **412**, **423** and **431** define an anticlockwise cycle.

All three-state machines relevant to this invention are listed in FIG. **9**. The six state-to-state transitions **412**, **421**, **423**, **432**, **431** and **413** are named T**12**, T**21**, T**23**, T**32**, T**31** and T**13**, respectively. Each row defines one possible three-state machine. For each state-to-state transition, the existence or not of that transition is indicated. The three-state machine defined by row **900** is a preferred embodiment that corresponds to three-state machine **400** of FIG. **4**. The three-state machines defined by rows **901**, **902**, **903**, **904**, **905**, and **906** belong to the class of three-state machine **500** of FIG. **5**. The three-state machines defined by rows **907**, **908**, **909**, **910**, **911**, and **912** belong to the class of three-state machine **600** of FIG. **6**. The three-state machines defined by rows **913**, **914**, and **915** belong to the class of three-state machine **700** of FIG. **7**. The three-state machines defined by rows **916** and **917** belong to the class of three-state machine **800** of FIG. **8**.

At any one time, each slave is at one and only one of the three states **401**, **402**, or **403**. Accordingly, there are three sets of slaves; those are state **401**, those are state **402**, and those at state **403**. State transitions are equivalent to movement between those three sets. This view of operating over sets is illustrated in FIG. **10**. Set **1001**, named E**1**, contains as elements all slaves **1010** that are at state **401**. Set **1002**, named E**2**, contains as elements all slaves **1020** that are at state **402**. Set **1003**, named E**3**, contains as elements all slaves **1030** that are at state **403**. There are six base commands, irrespective of the condition they specify, for effecting movement of elements between those three sets. Command **1012**, named T**12**, moves elements from set **1001** to set **1002**. Command **1021**, named T**21**, moves elements from set **1002** to set **1001**. Command **1023**, named T**23**, moves elements from set **1002** to set **1003**. Command **1032**, named T**32**, moves elements from set **1003** to set **1002**. Command **1031**, named T**31**, moves elements from set **1003** to set **1001**. Command **1013**, named T**13**, moves elements from set **1001** to set **1003**.

The simplest form of command is illustrated in FIG. **11**. Command **1110** comprises three parameters. First, the "from" state **1101**; second, the "to" state **1102**; and third, the primitive condition **1112** which must be satisfied for the transition to happen. Those three parameters are named Si, Sj, and s, respectively in the figure. Si, the "from" state **1101**, and Sj, the "to" state **1102**, can be any of the three states **401**, **402** or **403**, except that Si and Sj are not equal.

The primitive condition may take many forms depending on the overall capabilities of the slaves and the purposes that underlie the need for selecting subsets of slaves. Such primitive conditions could take the form of equality testing or numerical comparisons. Even though a single command broadcast from the master to the slave can only specify a single primitive condition, arbitrarily complex conditions are realized by a sequence of these primitive commands. In a preferred embodiment, an arbitrarily complex condition is described by a logical equation over primitive conditions. For example, the complex condition A*B+~A*~B, where A and B are primitive conditions, "*" is the binary logical operator AND, "+" the binary logical operator OR, and "~" the unary logical operator NOT. Negated primitive conditions, such as ~A, are assumed to be primitive conditions.

It is convenient for expositional purposes to textually represent command **1110**. A simple syntax used herein is to write the command as Tij(s), where i and j are **1**, **2**, or **3**, corresponding to states **401**, **402**, **403**, respectively, and s is the condition to be satisfied. The prefix T, for transition, is purely cosmetic. For example, T**31**(~A) represents a command to move all slaves that are at the third state and which do not satisfy A, to the first state, while T**23**(**1**) represents a command to move all slaves that are at the second state to the third state unconditionally.

The six possible transitions **412**, **413**, **421**, **423**, **431** and **432** for some condition s, can thus be written has T**12**(s), T**13**(s), T**21**(s), T**23**(s), T**31**(s) and T**32**(s), respectively. Any command thus involves only two of the three states of a three-state machine and only one of the six possible transitions. The pair of states **1100** in FIG. **11** and the single transition between them defines the scope of a single command. Only slaves that are at state Si, that is the "from" state **1101**, of the pair of states **1100** are allowed to transition, and those that do transition will do it to state Sj, that is the "to" state **1102**. Condition s is tested by each slave that is at state Si. Those for which the condition is satisfied will switch to state Sj. These semantics are expressed by the two logical expressions

$$Ei=Ei*\sim s$$

$$Ej=Ej+Ei*s$$

The first expression states that the set Ei, of all slaves that are at state Si, is decremented by the number of slaves that move from Si to Sj. That is expressed in the form of a logical AND between the previous value of set Ei and the virtual set of all slaves in Ei that did not satisfy condition s. Concurrently, the second expression states that the set Ej, of all slaves that are at state Sj, is augmented by the number of slaves that have moved from Si to Sj. That is expressed in the form of a logical OR between the previous value of set Ej and the virtual set of all slaves in Ei that satisfy condition s, the latter expressed by a logical AND between the previous value of set Ei and the virtual set of all slaves in Ei that satisfy condition s.

Since a command **1110** is broadcast to all slaves and they receive and operate on it concurrently, the command is essentially an operation over sets. The command Tij(s) effectively moves elements from a set Ei, of all slaves at state Si, to a set Ej, of all slaves at state Sj. Therefore sets E**1**, E**2** and E**3** are associated to states S**1**, S**2** and S**3**, respectively. Those two notions, sets and states, are for the purpose of this invention functionally equivalent. Reference herein to states S**1**, S**2** and S**3** imply sets E**1**, E**2** and E**3**, and vice versa, respectively.

FIG. **12** illustrates by means of Venn diagrams a simple non limiting example of one of such command. Sets **1001**, **1002** and **1003** correspond to slaves that are in the first, second and third state, respectively. For this example, set **1001** initially contains all slaves, while sets **1002** and **1003** are empty. Three testable primitive conditions A, B and C are defined. Each one of these three primitive conditions defines a virtual subset **1205**, **1206** and **1207**, respectively. The negation of each primitive condition, namely ~A, ~B and ~C, defines three complementary virtual subsets to **1205**, **1206** and **1207**, respectively. Those virtual subsets may or may not have slaves in common. In the figure we assume the most difficult case where virtual subsets **1205**, **1206** and **1207** intersect each other. Command T**12**(A) is broadcast, which forces all slaves in virtual subset **1205** in set **1001** to move from set **1001** to set **1002**. Therefore, after the command is executed by all slaves in this situation, the right half of the figure shows that set **1001** represents the ~A condition, set **1002**, the A condition, and set **1003** is empty.

Note that other Figures in this disclosure that are Venn diagrams have there sets numbered in the same manner as FIG. **12** but that these number are not shown for clarity.

Another example is shown in FIG. **13**. The starting configuration is the same as in FIG. **12**. However, the command T**12**(~A) is broadcast instead. As the right half of the figure indicates, after the command is executed by all slaves in this situation, set **1001** represents the A condition, set **1002**, the ~A condition, and set **1003** is empty.

A condition that is the product of two or more primitive conditions is obtained by two or more commands. An AND condition A*B, for example, can be obtained by broadcasting two or more commands. First, T**12**(A), then T**21**(~B). Starting from the same initial configuration as used in FIG. **12**, execution of T**12**(A) is shown in FIG. **12**. Execution of command T**21**(~B) on that resulting configuration is shown in FIG. **14**. As the right half of the figure indicates, after the two commands are executed in succession by all slaves in this situation, set **1001** represents the ~(A*B) condition, set **1002**, the A*B condition, and set **1003** is empty. Therefore, set **1002** represents an AND condition, while set **1001** represents the complementary NAND condition.

A condition that is the sum of two or more primitive conditions is obtained by two or more commands. An OR condition A+B, for example, can be obtained by broadcasting two commands. First, T**12**(A), then T**12**(B). Starting from the same initial configuration as used in FIG. **12**, execution of T**12**(A) is shown in FIG. **12**. Execution of command T**12**(B) on that resulting configuration is shown in FIG. **15**. As the figure indicates, after the two commands are executed in succession by all slaves in this situation, set **1001** represents the ~(A+B) condition, set **1002**, the A+B condition, and set **1003** is empty. Therefore, set **1002** represents an OR condition, while set **1001** represents the complementary NOR condition.

The previous two examples involve only two of the possible three sets. Some conditions require the use of all three sets. The only two conditions involving two primitive conditions A and B that require all three sets are the EXCLUSIVE-OR, A*~B+~A*B, and its complement the EQUIVALENCE, A*B+~A*~B. (See discussion of left nested expressions below.) The EXCLUSIVE-OR can be obtained by broadcasting a sequence of three commands. T**12**(~A), T**13**(B) and T**21**(B). Execution of the three commands is shown in FIGS. **13**, **16**, and **17**. This sequence is more compactly represented in tabular form as done in FIG. **18**. Row **1800** of the table is the initial state of the three sets. In this example, set **1001** has all slaves and sets **1002** and **1003** are empty. Rows **1801**, **1802** and **1803** show the result of executing commands T**12**(~A), T**13**(B) and T**21**(B), respectively, and the resulting conditions expressed by each of the three sets after each command is executed. As FIG. **18** shows, after the three commands are executed in succession by all slaves in this situation, set **1001** represents the A*~B+~A*B condition, set **1002**, the ~A*~B condition, and set **1003**, the A*B condition. As is, the slaves that satisfy the EXCLUSIVE-OR condition ended up in set **1001**. If the goal had been to place those slaves in set **1002** instead, a different sequence of commands would be broadcast. That sequence is shown in FIG. **19**. Rows **1900**, **1901**, **1902** and **1903** of the table show the initial and succeeding conditions obtained in each set during the execution of the sequence of commands T**12**(A), T**23**(B) and T**12**(B). If the goal had been to place the EXCLUSIVE-OR in set **1003** instead, a different sequence of commands would be broadcast. That sequence is shown in FIG. **20**. Rows **2000**, **2001**, **2002** and **2003** of the table show the initial and succeeding conditions obtained in each set during the execution of the sequence of commands T**13**(A), T**32**(B) and T**13**(B).

The present invention teaches several methods for generating a command sequence necessary to select slaves that satisfy an arbitrarily complex condition involving any number of primitive conditions. Before describing the most general methods, the invention first teaches a few important methods aimed at certain types of complex conditions. Non limiting examples of command sequences are given in the columns numbered **1810** (in FIG. **18**), **1910** (in FIG. **19**), **2010** (in FIG. **20**), **2610** (in FIG. **26**), **2710** (in FIG. **27**), **2910** (in FIG. **29**), **3010** (in FIG. **30**), **3210** (in FIG. **32**), **3310** (in FIG. **33**), **3510** (in FIG. **35**), **3610** (in FIG. **36**), **3910** (in FIG. **39**), and **4010** (in FIG. **40**).

Each command, T, sent from the master to one or more slaves, has a single primitive condition, ci, that is one of any number of arbitrary primitive conditions. The command, T, addresses some information stored on each of the slaves and causes the respective slave to compare its stored information with the primitive condition.

The first of those is a condition expressed by the product of two or more primitive conditions. This is the general AND condition and the method for handling that kind of condition is shown in FIG. **21**. Method **2100** takes as input **2101** two different sets Ej and Ek of the possible three sets E**1**, E**2** and E**3**, and an AND of N primitive conditions, that is c**1***c**2*** . . . *cN. Method **2100** outputs a configuration **2102** whereby all slaves in set Ej that satisfied the aforementioned AND condition have moved to set Ek. If set Ek was not empty to start with, a side effect of the method is that all slaves originally in Ek that did not satisfy the reduced condition c**2*** . . . *cN have moved to set Ej. That can be represented mathematically as

$$Ej=Ej*\sim(c1*c2* \ldots *cN)+Ek*\sim(c2*c3* \ldots *cN)$$

$$Ek=Ej*(c1*c2* \ldots *cN)+Ek*(c2*c3* \ldots *cN)$$

Method **2100** accomplishes this by generating a sequence of N commands. In step **2110**, a command is issued that

causes all slaves in set Ej that satisfy first primitive condition c1 to move to set Ek. This command is written as Tjk(c1). The state transitions effected can be mathematically represented as

$$Ej=Ej*\sim C1$$

$$Ek=Ek+Ej*c1$$

Step **2120** controls the iteration over all remaining primitive conditions that make up the input AND condition **2101**. For each primitive condition ci, where i varies from 2 to N, step **2121** issues a command that causes all slaves in set Ek that do not satisfy primitive condition ci to move to set Ej. This command is written as Tkj(~ci). The state transitions effected can be mathematically represented as

$$Ej=Ej+Ek*\sim c1$$

$$Ek=Ek*c1$$

The iteration ends after the last primitive condition, cN, has been processed by step **2121**. That terminates the method.

In a preferred embodiment, Ek begins as a null set so that the slaves that are found in Ek at the end of method **2100** are exactly those that satisfy the AND condition.

The second is a condition expressed by the negation of a product of two or more primitive conditions. This is the general NAND condition and the method for handling that kind of condition is shown in FIG. **22**. Method **2200** takes an input **2201** two different sets Ej and Ek of the possible three sets E1, E2 and E3, and a NAND of N primitive conditions, that is, ~(c1*c2* . . . *cN). Method **2200** outputs a configuration **2202** whereby all slaves in set Ej that satisfy the aforementioned NAND condition are moved to set Ek. That can be represented mathematically as

$$Ej=Ej*(c1*c2* \ldots *cN)$$

$$Ek=Ek+Ej*\sim(c1*c2* \ldots *cN)$$

Method **2200** accomplishes this by generating a sequence of N commands. The main step **2220** controls the iteration over all the primitive conditions that make up the input NAND condition. For each primitive condition ci, where i varies from 1 to N, step **2221** issues a command that causes all slaves in set Ej that do not satisfy primitive condition ci to move to set Ek. This command is written as Tjk(~ci). The state transitions effected can be mathematically represented as

$$Ej=Ej*c1$$

$$Ek=Ek+Ej*\sim c1$$

The iteration ends after the last primitive condition, cN, has been processed by step **2221**. That terminates the method.

In a preferred embodiment, Ek begins as a null set so that the slaves that are found in Ek at the end of method **2200** are exactly those that satisfy the NAND condition.

The third is a condition expressed by the sum of two or more primitive conditions. This is the general OR condition and the method for handling that kind of condition is shown in FIG. **23**. Method **2300** takes as input **2301** two different sets Ej and Ek of the possible three sets E1, E2 and E3, and an OR of N primitive conditions, that is, c1+c2+. . . +cN. Method **2300** outputs a configuration **2302** whereby all slaves in set Ej that satisfy the aforementioned OR condition are moved to set Ek. That can be represented mathematically as

$$Ej=Ej*\sim(c1+c2+ \ldots +cN)$$

$$Ej=Ek+Ej*(c1+c2+ \ldots +cN)$$

Method **2300** accomplishes this by generating a sequence of N commands. The main step **2320** controls the iteration over all the primitive conditions that make up the input OR condition. For each primitive condition ci, where i varies from 1 to N, step **2321** issues a command that causes all slaves in set Ej that satisfy primitive condition ci to move to set Ek. This command is written as Tjk(ci). The state transitions effected can be mathematically represented as

$$Ej=Ej*\sim c1$$

$$Ek=Ek+Ej*c1$$

The iteration ends after the last primitive condition, cN, has been processed by step **2321**. That terminates the method.

In a preferred embodiment, Ek begins as a null set so that the slaves that are found in Ek at the end of method **2300** are exactly those that satisfy the OR condition.

The fourth is a condition expressed by the negation of the sum of two or more primitive conditions. This is the general NOR condition and the method for handling that kind of condition is shown in FIG. **24**. Method **2400** takes as input **2401** two different sets Ej and Ek of the possible three sets E1, E2 and E3, and a NOR of N primitive conditions, that is, ~(c1+c2+. . . +cN). Method **2400** outputs a configuration **2402** whereby all slaves in set Ej that satisfy the aforementioned NOR condition are moved to set Ek. If set Ek was not empty to start with, a side effect of the method is that all slaves originally in Ek that satisfied the reduced condition c2+. . . +cN have moved to set Ej. That can be represented mathematically as

$$Ej=Ej*(c1+c2+ \ldots +cN)+Ek*(c2+c3+ \ldots +cN)$$

$$Ek=Ej*\sim(c1+c2+ \ldots +cN)+Ek*\sim(c2+c3+ \ldots +cN)$$

Method **2400** accomplishes this by generating a sequence of N commands. In step **2410**, a command is issued that causes all slaves in set Ej that do not satisfy first primitive condition c1 to move to set Ek. This command is written as Tjk(~c1). The state transitions effected can be mathematically represented as

$$Ej=Ej*c1$$

$$Ek=Ek+Ej*\sim c1$$

Step **2420** then controls the iteration over all the remaining primitive conditions that make up the input NOR condition. For each primitive condition ci, where i varies from 2 to N, step **2421** issues a command that causes all slaves in set Ek that satisfy primitive condition ci to move to set Ej. This command is written as Tkj(ci).

$$Ej=Ej+Ek*ci$$

$$Ek=Ek*\sim ci$$

The iterative step **2420** ends after the last primitive condition, cN, has been processed by step **2421**. That terminates the method.

In a preferred embodiment, Ek begins as a null set so that the slaves that are found in Ek at the end of method **2400** are exactly those that satisfy the NOR condition.

Methods **2100**, **2200**, **2300** and **2400** can be combined to handle arbitrarily complex conditions. The simplest such

combination is called canonical, because it is based on the well-known technique of expressing an arbitrarily complex condition in the form of sum-of-products, i.e., one or more ANDed primitive conditions that are ORed together. The canonical method works by computing each product term of the condition using two sets and accumulating, that is summing, the product terms into a third set.

FIG. **25** shows a method **2500** that moves slaves from the first to the second state and visa versa. When slaves are in the second state, it is possible to move them to the third state where all remaining commands in the command sequence are ignored. Method **2500** takes as input **2501** three different sets Ej, Ek, and El, that is, some permutation of sets E**1**, E**2** and E**3**, and a condition that is a sum of P product terms, that is, p**1**+p**2**+. . . +pP. Assuming, for simplicity, that set Ek is empty at the start of the method. Method **2500** outputs a configuration **2502** whereby all slaves in set Ej that satisfy the aforementioned sum-of-products condition are moved to set E**1**. That can be represented mathematically as

$$Ej=Ej*\sim(p1+p2+\ldots+pP)$$

$$Ek=0$$

$$El=El+Ej*(p1+p2+\ldots+pP)$$

Method **2500** accomplishes this by generating a sequence of commands. The main step **2520** controls the iteration over all the product terms that make up the input sum-of-products condition **2501**. For each product term pi, where i varies from 1 to P, first step **2421** issues a sequence of commands as defined by method **2100** that causes all slaves in set Ej that satisfy the product term pi to move to set Ek. Second step **2522** issues a command that causes all slaves in set Ek to move to set E**1**. The iteration ends after the last product term, pP, has been processed by steps **2521** and **2522**. That terminates the method.

In other words, the method **2500** creates each of the product terms in **2521** and in set Ek. After each product term, pi, is created, it is ORed with the previously accumulated product terms in set E**1**. That frees up set Ek in preparation for the next product term.

Applying the method **2500** to the EXCLUSIVE-OR condition ~A*B+A*~B, for example, results in the command sequence **2610** shown in FIG. **26**. The default initial configuration **2600** where all the slaves are in set E**1** is used. In this example, j=**1**, k=**2** and l=**3**. Commands T**12**(~A) and T**21**(~B), as per method **2100**, are used to move slaves that satisfy the product term ~A*B from set E**1** to set E**2**, as shown by rows **2601** and **2602**. Command T**23**(**1**) sums that product term from E**2** to E**3**, as shown in row **2603**. That is the basic cycle for one product term. Next, commands T**12**(A), T**21**(B) and T**23**(**1**) repeat the cycle to move slaves that satisfy the next product term A*~B first from set E**1** to set E**2** and second from set E**2** to set E**3**, as shown by rows **2604**, **2605** and **2606**. Six commands are generated by method **2500** for the EXCLUSIVE-OR condition, two of those on account of two product terms and the other four on account of four primitive conditions present over all product terms. Contrast this with the three commands generated by any the hand crafted solution of FIGS. **18**, **19** and **20**. While the canonical method is easy to compute, the command sequences it generates are not minimal in general.

An arbitrarily complex condition can be written in the form of a sum-of-products. A canonical method for generating a command sequence for a sum-of-products is to use first and second states to calculate product terms and to use the third state to accumulate the product terms. The canonical method does not yield the shortest command sequence but is easy to compute.

When a complex condition is put in sum-of-product form, the products do not have to be expanded so that each contains all primitive conditions used in the complex condition. For an example which includes three primitive conditions A, B, and C, the complex condition A*~B*C+A*B*C+~A*B*C+A*B*C can be minimized to A*C+B*C and still be considered a sum-of-products for the purpose of method **2500** and other methods described hereunder. Such a minimization represents a significant reduction in commands required. While the fully expanded condition above would require sixteen commands (four product terms and twelve primitive condition appearances), the corresponding minimized sum-of-products requires six commands (two product terms and four primitive condition appearances), when both the command sequences are generated through the canonical method **2500**. The six commands solution **2710** is shown in FIG. **27** and not surprisingly mimics the command sequence **2610** of FIG. **26**. Again a default initial configuration **2700** where all slaves are in set E**1** is used. In this example, j=**1**, k=**2** and l=**3**. Command T**12**(A) transfers from set E**1** to set E**2** the slaves in set E**1** that satisfy primitive condition A. Row **2701** indicates that set E**1** contains the slaves that satisfy condition ~A; set E**2**, condition A; and set E**3** is empty. Command T**21**(~C) transfers from set E**2** to set E**1** the slaves in set E**2** that do not satisfy primitive condition C. Row **2702** indicates that set E**1** represents condition ~A+A*~C, which is equivalent to ~A+~C; set E**2**, condition A*C; and set E**3** is empty. Command T**23**(l) transfers from set E**2** to set E**3** all slaves in set E**2**. Row **2703** indicates the accumulation of product term A*C into set E**3**. Command T**12**(B) transfers from set E**1** to set E**23** the slaves in set E**1** that satisfy primitive condition B. Row **2704** indicates that set E**1** represents the condition (~A+~C)*~B, which is equivalent to condition ~A*~B+~B*~C; and set E**2**, condition (~A+~C)*B, which is equivalent to ~A*B+B*~C. Command T**21** (~C) transfers from set E**2** to set E**1** the slaves in set E**2** that do not satisfy primitive condition C. Row **2705** indicates that set E**1** represents condition ~A*~B+~B*~C+(~A*B+B*~C)*~C, which reduces to ~A*~B+~B*~C+~A*B*~C+B*~C, then to ~A*~B+~B*~C+B*~C, then to ~A*~B+~C; and set E**2** represents condition (~A*B+B*~C)*C, which is equivalent to ~A*B*C. Command T**23**(**1**) transfers from set E**2** to set E**3** all slaves in set E**2**. Row **2706** indicates that set E**3** represents the condition A*C+~A*B*C, which reduces to (A+~A*B)*C, then to (A+B)*C, then to A*C+B*C, which is the sum-of-product condition that needed to be satisfied. Applying method **2500** to a minimized sum-of-products, as in this example, will generate a command sequence that is certainly shorter than a full-expanded sum-of-products, but is still not necessarily minimal.

Characteristic of method **2500** is that the first set Ej serves as the main repository of slaves and will end up containing the slaves originally in set Ej that do not satisfy the sum-of-products condition. Second set Ek serves to build each product term. Third set E**1** serves to sum the product terms and will end up containing the slaves originally in set Ej that satisfy the sum-of-products condition. Therefore, for method **2500**, each of the three sets has a uniquely defined role.

This does not need to be the case and one can create variations of method **2500** where the roles alternate. The method shown in FIG. **28** is one such variation. Method **2800** differs from method **2500** in that the roles of states Ej (state **1**) and Ek (state **2**) alternate, i.e. states **1** and **2** reverse

roles. Product terms are build alternatively in states Ek and Ej: first in Ek, then in Ej, then back in Ek, and so on; in other words, odd product terms—first, third, fifth, etc.—are built in set Ek, while even product terms—second, fourth, sixth, etc.—are built in set Ej. Method **2800** is similar to method **2500** in that the role of state E**1** remains the same. Method **2800** takes as input **2801** the same input as does method **2500**. Method **2800** takes as input **2801** the same input as does method **2500**. Method **2800** outputs a configuration whereby all slaves in Ej that satisfy the sum-of-products condition are moved to set E**1**, and either set Ej or Ek will be empty depending on the number of product terms. If the condition has an even number of product terms, Ek will be empty; otherwise, Ej will be empty. That can be represented mathematically as

$El=El+Ej^{*}(p1+p2+\ldots+pP)$

(if P is even) $Ej=Ej^{*}\sim(p1+p2+\ldots+pP)$

(if P is odd) $Ej=0$

(if P is even) $Ek=0$

(if P is odd) $Ek=Ej^{*}\sim(p1+p2+\ldots+pP)$

The main step **2820** controls the iteration over all the product terms that make up the sum-of-products condition **2801**. For each product term pi, where i varies from 1 to P, step **2830** tests whether i is odd or even. If i is odd, steps **2831** and **2833** are executed for product term pi. If i is even, steps **2832** and **2834** are executed for product term pi. Step **2831** issues a sequences of commands defined by NAND method **2200** that causes all slaves in set Ej that do not satisfy product term pi to move to set Ek. Step **2833** issues a command that causes all slaves in set Ej to move to set El. Similarly, step **2832** issues a sequence of commands defined by NAND method **2200** that causes all slaves in set Ek that do not satisfy product term pi to move to set Ej. Step **2834** issues a command that causes all slaves in set Ek to move to set El. The iteration ends after the last product term, pP, has been processed by either steps **2831** and **2833** (odd P case), or steps **2832** and **2834** (even P case). That terminates the method.

If method **2800** is used on the EXCLUSIVE-OR condition ~A*B+A*~B, the sequence of commands **2910** shown in FIG. **29** results. The default initial configuration **2900** where all slaves are in set E**1** is used. In this example, j=**1**, k=**2** and l=**3**. Rows **2901** and **2902** correspond to the application of method **2200** from E**1** to E**2** to the product term ~A*B. Row **2903** is the accumulation of that product term in E**3**. Rows **2904** and **2905** correspond to the application of method **2200** from E**2** to E**1** to the product term A*~B. Row **2906** is the accumulation of that product term in E**3**. Because the number of product term is even in this case, E**2** is empty at the end.

A similar sequence of commands results if method **2800** is used on the condition A*C+B*C. The sequence of commands **3010** is shown in FIG. **30**. The default initial configuration **3000** where all slaves are in set E**1** is used. In this example, j=**1**, k=**2** and l=**3**. Rows **3001** and **3002** correspond to the application of method **2200** from E**1** to E**2** to the product term A*C. Row **3003** is the accumulation of that product term in E**3**. Rows **3004** and **3005** correspond to the application of method **2200** from E**2** to E**1** to the product term B*C. Row **3006** is the accumulation of that product term in E**3**. Because the number of product term is even in this case, E**2** is empty at the end.

The method shown in FIG. **31** differs from methods **2500** and **2800** in that both the building of product terms and the accumulation of product terms alternates between two sets. Specifically, the roles of set Ek (state **2**) and E**1** (state **3**) reverse. With method **3100** product terms are either computed from set Ej to set Ek or from set Ej to set El. While for methods **2500** and **2800** summing was done by adding the latest product term into the previous accumulation, with method **3100** the previous accumulation is added to the latest product term. Accordingly, when the latest product term is built in set Ek, accumulation is from El to Ek, and when the latest product term is built in set El, accumulation is from Ek to El. Method **3100** takes as input **3101** the same input as do methods **2500** and **2800**. Method **3100** outputs a configuration whereby all slaves in Ej that satisfy the sum-of-products condition are moved to either set Ek or El depending on the number of product terms. If the condition has an even number of product terms, El will contains the desired slaves; otherwise, Ek will. That can be represented mathematically as

$Ej=Ej^{*}\sim(p1+p2+\ldots+pP)$

(if P is odd) $Ek=El+Ej^{*}(p1+p2+\ldots+pP)$

(if P is even) $Ek=0$

(if P is odd) $El=0$

(if P is even) $El=El+Ej^{*}(p1+p2+\ldots+pP)$

The main step **3120** controls the iteration over all the product terms that make up the sum-of-products condition **3101**. For each product term pi, where i varies from 1 to P, step **3130** tests whether i is odd or even. If i is odd, steps **3131** and **3133** are executed for product term pi. If i is even, steps **3132** and **3134** are executed for product term pi. Step **3131** issues a sequence of commands defined by AND method **2100** that causes all slaves in set Ej that satisfy product term pi to move to set Ek. Step **3133** issues a command that causes all slaves in set El to move to set Ek. Similarly, step **3132** issues a sequence of commands defined by AND method **2100** that causes all slaves in set Ej that satisfy product term pi to move to set El. Step **3134** issues a command that causes all slaves in set Ek to move to set El. The iterative step **3120** ends after the last product term, pP, has been processed by either steps **3131** and **3133** (odd P case), or steps **3132** and **3134** (even P case). That terminates the method.

If method **3100** is used on the EXCLUSIVE-OR condition ~A*B+A*~B, the sequence of commands **3210** shown in FIG. **32** results. The default initial configuration **3200** where all slaves are in set El is used. In this example j=**1**, k=**2**, and l=**3**. Rows **3201** and **3202** correspond to the application of method **2100** from E**1** to E**2** to the product term ~A*B. Row **3203** is the accumulation of E**3** into that product term. Because E**3** is empty, this command is unnecessary but is included as part of the normal cycle. Rows **3204** and **3205** correspond to the application of method **2100** from E**1** to E**3** to the product term A*~B. Row **3206** is the accumulation of E**2** into that product term. Because the number of product term is even in this case, E**3** contains the desired set of slaves.

A similar sequence of commands results if method **3100** is used on the condition A*C+B*C. The sequence of commands **3310** is shown in FIG. **33**. The default initial configuration **3300** where all slaves are in set E**1** is used. In this example, j=**1**, k=**2** and l=**3**. Rows **3301** and **3302** correspond to the application of method **2100** from E**1** to E**2** to the product term A*C. Row **3303** is the accumulation of E**3** into that product term. Because E**3** is empty, this command is

unnecessary but is included as part of the normal cycle. Rows **3304** and **3305** correspond to the application of method **2100** from E**1** to E**3** to the product term B*C. Row **3306** is the accumulation of E**2** into that product term. Because the number of product term is even in this case, E**3** contains the desired set of slaves.

The method shown in FIG. **34** differs from methods **2500**, **2800** and **3100** in that the building of product terms and the accumulation of those products terms rotates among three sets, i.e., the states cycle roles. Method **3400** takes as input **3401** the same input as do methods **2500**, **2800** and **3100**. Method **3400** outputs a configuration whereby all slaves in set Ej that satisfy the sum-of-products condition are moved to set Ej, Ek or El depending on the number of product terms. If the number of product terms modulo **3** is one, that is, 1, 4, 7, etc., set Ej will end up containing the desired set of slaves. If the number of product terms modulo **3** is two, that is 2, 5, 8, etc., set Ek will end up containing the desired set of slaves. If the number of product terms modulo **3** is zero, that is, 3, 6, 9, etc., set E**1** will end up containing the desired set of slaves. That can be represented mathematically as

(if (P mod **3**)=1) *Ej=El+p***1**+*p***2**+. . . +*pP, El*=0

(if (P mod **3**)=2) *Ek=El+p***1**+*p***2**+. . . +*pP, Ej*=0

(if (P mod **3**)=0) *El=El+p***1**+*p***2**+. . . +*pP, Ek=0*

The main step **3410** controls the iteration over all product terms that make up the sum-of-products condition **3401**. For each product term pi, where i varies from 1 to P, step **3420** tests whether i mod **3** is one, two or zero. If one, steps **3421** and **3424** are executed for product term pi. If two, steps **3422** and **3425** are executed for product term pi. If zero, steps **3423** and **3426** are executed for product term pi. Step **3421** issues a sequence of commands defined by NAND method **2200** that causes all slaves in set Ej that do not satisfy product term pi to move to set Ek. Step **3424** issues a command that causes all slaves in set El to move to set Ej. Similarly, step **3422** issues a sequence of commands defined by NAND method **2200** that causes all slaves in set Ek that do not satisfy product term pi to move to set El. Step **3425** issues a command that causes all slaves in set Ej to move to set Ek. Similarly, step **3423** issues a sequence of commands defined by NAND method **2200** that causes all slaves in set E**1** that do not satisfy product term pi to move to set Ej. Step **3426** issues a command that causes all slaves in set Ek to move to set El. The iterative step **3410** ends after the last product term, pP, has been processed. That terminates the method.

If method **3400** is used on the EXCLUSIVE-OR condition ~A*B+A*~B, the sequence of commands **3510** shown in FIG. **35** results. The default initial configuration **3500** where all slaves are in set El is used. In this example j=**32 1**, k=**2** and l=**3**. Rows **3501** and **3502** correspond to the application of method **2200** from E**1** to E**2** to the negated product term ~(~A*B). Row **3503** is the accumulation of E**3** into E**1**. Because E**3** is empty, this command is unnecessary but is included as part of the normal cycle. Rows **3504** and **3505** correspond to the application of method **2200** from E**2** to E**3** to the negated product term ~A*~B). Row **3506** is the accumulation of E**1** into E**2**. Because the number of product terms modulo three is two in this case, E**2** contains the desired set of slaves.

A similar sequence of commands results if method **3500** is used on the condition A*C+B*C. The sequence of commands **3610** is shown in FIG. **36**. The default initial configuration **3600** where all slaves are in set E**1** is used. In this example, j=l, k=**2** and l=**3**. Rows **3601** and **3602** correspond to the application of method **2200** from E**1** to E**2** to the negated product term ~(A*C). Row **3603** is the accumulation of E**3** into E**1**. Because E**3** is empty, this command is unnecessary but is included as part of the normal cycle. Rows **3604** and **3605** correspond to the application of method **2200** from E**2** to E**3** to the negated product term ~(B*C). Row **3606** is the accumulation of E**1** into E**2**. Because the number of product terms modulo three is two in this case, E**2** contains the desired set of slaves.

Methods **2500**, **2800**, **3100** and **3400** do not in general generate a minimal sequence of commands for a given arbitrarily complex condition expressed in sum-of-products form. A shorter command sequence can be obtained when an arbitrarily complex condition can be written by an expression that can be generated by the following grammar:

ln-expression: (ln-expression)*primitive_condition

ln-expression: ln-expression+primitive_condition

ln-expression: primitive_condition

where ln-expression is the name given to this kind of expression, namely, left-nesting expression. A ln-expression can be written as, ((. . . (((c**1**) op**2** c**2**) op**3** c**3**) . . .) opN cN), where c**1**, c**2**, . . . , cN are primitive conditions and op**2**, op**3**, . . . , opN are either * (AND) or +(OR) binary operators. The ln-expression as written above is more heavily parenthetically bracketed than necessary and some parenthesis may be deleted as long as the logic is preserved. Left-nesting expressions can be executed using only two of the three states of the three-state machine. An arbitrarily complex condition such as A*B+A*C can be expressed according to the grammar as (B+C)*A. The aforementioned canonical method over the former, sum-of-products, expression requires six commands to execute and uses three states. The latter, left-nesting, expression can be computed with only three commands and uses only two states. Not every arbitrarily complex conditions can be expressed by a single left-nested expression, but any complex condition can be expressed by a sum of left-nested expressions, which requires fewer commands than the canonical sum-of-products form. For example, the condition A*B+A*C+~A*~B+~A*~C can be written as a sum of two left-nested expressions: (B+C)*A+(~B+~C)*~A; the former requires twelve commands, while the latter only eight. As with the canonical sum-of-products method, which uses the third state to accumulate products, the method for executing sum-of-left-nested-expressions uses the third state to accumulate left-nested expressions.

As mentioned above, the present three-selection-state machine is capable of isolating or selecting slaves that satisfy any possible condition expressed by a left-nested expression. Specifically, the invention is necessary and sufficient to isolate and select slaves satisfying those conditions that are expressed by a sum-of-left-nested-expressions. The invention enables this capability because a separate condition (or set of conditions), each corresponding to a set of slaves, can be isolated in any one of the three states at any given time. Therefore, operations on two sets of conditions, in two of the respective states, can be performed without affecting or being affected by the conditions held in the third state. Specific instances of left-nested-expressions handled by the invention are now presented.

The method for computing the sequence of commands necessary to transfer from a set Ej to a set Ek slaves in set

Ej that satisfy a condition given as a ln-expression is shown in FIG. **37**. Method **3700** takes as input **3701** two different sets Ej and Ek of the possible three sets E**1**, E**2** and E**3**, and a ln-expression, ((. . . (((c**1**) op**2** c**2**) op**3** c**3**) . . .) opN cN). Method **3700** outputs a configuration **3702** whereby all slaves in set Ej that satisfy the ln-expression of input **3701** are moved to set Ek. If set Ek was not empty to start with, a side effect of the method is that all slaves originally in set Ek that did not satisfy the reduced condition (. . . ((cM) op . . .) . . . opN cN)) where M is such that opM is the first * (AND) operator in the ln-expression, have moved to set Ej. That can be represented mathematically as

$$Ej=Ek*\sim((cm\ op\ \ldots)opN\ cN)+Ej*\sim(c1\ op\ \ldots)\ opN\ cN)$$

$$Ek=Ek*\ ((cm\ op\ \ldots)\ opN\ cN)+Ej*\ ((c1\ op\ \ldots)\ opN\ cN)$$

where m such that op**2**, op**3**, . . . , opm−**1**=OR and opm=AND

Method **3700** begins with step **3710**, which issues a command that causes all slaves in set Ej that satisfy the leftmost (first) primitive condition c**1** to move to set Ek. Step **3720** controls the iteration over the binary operators and attendant right operands, from the leftmost to the rightmost, that is, from op**2** to opN and their attendant c**2** to cN. For each operator opi, where i varies from 2 to N, step **3730** tests which binary operator is opi. If opi is the AND operator *, step **3731** is executed; otherwise, opi is the OR operator+, in which case step **3732** is executed. Step **3731** issues a command that causes all slaves in set Ek that do not satisfy primitive condition ci to move to set Ej. Step **3732** issues a command that causes all slaves in set Ej that satisfy primitive condition ci to move to set Ek. After the last iteration, over opN and cN, step **3720** terminates the iteration. That terminates the method.

An important variation of method **3700** is shown in FIG. **38**. Method **3800** computes the sequence of commands necessary to transfer from a set Ej to a set Ek slaves in set Ej that satisfy a condition given as the negation of a ln-expression. Method **3800** takes an input **3801** two different sets Ej and Ek of the possible three sets E**1**, E**2** and E**3**, and a condition in the form of a negated ln-expression, ~((. . . (((c**1**) op**2** c**2**) op**3** c**3**) . . .) opN cN). Method **3800** outputs a configuration **3802** whereby all slaves in set Ej that satisfy the negated ln-expression of input **3801** are moved to set Ek. If set Ek was not empty to start with, a side effect of the method is that all slaves originally in set Ek that did not satisfy the reduced condition ~(. . . ((cM) op . . .) . . . opN cN), where M is such that opM is the first+(OR) operator in the ln-expression, have moved to set Ej. That can be represented mathematically as

$$Ej=Ek*((cm\ op\ \ldots)\ opN\ cN)+Ej*\ ((cl\ op\ \ldots)\ opN\ cN)$$

$$Ek=Ek*\sim((cm\ op\ \ldots)\ opN\ cN)+Ej*\sim((cl\ op\ \ldots)\ opN\ cN)$$

where m such that op**2**, op**3**, . . . , opm−**1**=AND and opm=OR

Method **3800** begins with step **3810**, which issues a command that causes all slaves in set Ej that do not satisfy the leftmost (first) primitive condition c**1** to move to set Ek. Step **3820** controls the iteration over the binary operators and attendant right operands, from the leftmost to the rightmost, that is, from op**2** to opN and their attendant c**2** to cN. For each operator opi, where i varies from 2 to N, step **3830** tests which binary operator is opi. If opi is the OR operator +, step **3831** is executed; otherwise, opi is the AND operator *, in which case step **3832** is executed. Step **3831** issues a command that causes all slaves in set Ek that satisfy primitive condition ci to move to set Ej. Step **3832** issues a command that causes all slaves in set Ej that do not satisfy primitive condition ci to move to set Ek. After the last iteration, over opN and cN, step **3820** terminates. That terminates the method.

By expressing the minimized sum-of-products condition A*C+B*C, used in previous examples, as an ln-expression (A+B)*C, either method **3700** or **3800** can be used to generate a sequence of commands that is shorter than the sequence generated by method **2500**, **2800**, **3100** or **3400**. The latter sequence is six commands long, as shown in FIGS. **27**, **30**, **33**, and **36**. Both methods **3700** and **3800** generate a sequence that is three commands long. The example sequence **3910** generated by method **3700** is shown in FIG. **39**. The example sequence **4010** generated by method **3800** is shown in FIG. **40**. They both start with the default initial configuration where all slaves are in set E**1**, as shown in rows **3900** and **4000**. In both examples j=**1**, k=**2** and l=**3**. Method **3700** generates the sequence T**12**(A), T**12**(B) and T**21**(~C). Commands T**12**(A) and T**12**(B) put the partial condition (A+B) in set E**2** as shown in rows **3901** and **3902**. Command T**21**(~C) results in the desired condition (A+B)*C in set E**2** as shown in row **3903**. Method **3800** generates the sequence T**12**(~A), T**21**(B) and T**12**(~C). Commands T**12**(~A) and T**21**(B) put the partial condition (A+B) in set E**1** as shown in rows **4001** and **4002**. Command T**12**(~C) results in the desired condition (A+B)*C in set E**1** as shown in row **4003**.

Note from the method descriptions of FIGS. **37** and **38**, and the examples of FIGS. **39** and **40**, that the third set E**3** is not used. It is an important property of conditions written as a single ln-expression, that they require only two of the three states of a three-state machine. This property permits the use of the third state, that is, of set E**3**, as an accumulator of ln-expressions. While not all arbitrarily complex conditions can be expressed by a single ln-expression, any arbitrary complex condition can be expressed by a sum-of-ln-expressions. It is possible therefore to recode methods **2500**, **2800**, **3100** and **3400** to work on sum-on-ln-expressions.

Method **2500** is recoded in FIG. **41** for sum-of-ln-expressions. Method **4100** takes an input **4101** three sets Ej, Ek and El, that is, some permutation of sets E**1**, E**2** and E**3**, and a condition written as a sum-of-ln-expressions, n**1**+n**2**+. . . +nN. Assuming, for simplicity, that set Ek is empty at the start of the method. Method **4100** outputs a configuration **4102** whereby all slaves in set Ej that satisfy the sum-of-ln-expressions condition **4101** are moved to set E**1**. That can be represented mathematically as

$$Ej=Ej*\sim(n1+n2+\ldots+nN)$$

$$Ek=0$$

$$El=El+Ej*(n1+n2+\ldots+nN)$$

The main step **4120** controls the iteration over all ln-expressions of condition **4101**. For each ln-expression ni, where i varies from 1 to N, steps **4121** and **4122** are executed in that order. Step **4121** issues a sequence of commands defined by method **3700** that causes all slaves in set Ej that satisfy the ln-expression ni to move to set Ek. Step **4122** issues a command that causes all slaves in set Ek to move to set El. Iteration ends after the last ln-expression, nN, has been processed. That terminates the method.

Method **2800** is recoded in FIG. **42** for sum-of-ln-expressions. Method **4200** takes as input **4201** the same input as does method **4100**. Method **4200** outputs a con-

figuration **4202** whereby all slaves in Ej that satisfy the sum-of-ln-expressions condition **4201** are moved to set El, and either set Ej or set Ek will be empty depending on the number of ln-expressions. If the condition has an even number of ln-expressions, set Ek will be empty; otherwise, set Ej will be empty. That can be represented mathematically as

$$El=El+Ej^*(n1+n2+\ldots+nN)$$

(if N is even) $Ej=Ej^*\sim(n1+n2+\ldots+nN), Ek=0$

(if N is odd) $Ek=Ej^*\sim(n1+n2+\ldots+nN), Ej=0$

The main step **4220** controls the iteration over all ln-expressions of condition **4201**. For each ln-expression ni, where i varies from 1 to N, step **4230** tests whether i is odd or even. If i is odd, steps **4231** and **4233** are executed for ln-expression ni. If i is even, steps **4232** and **4234** are executed for ln-expression ni. Step **4231** issues a sequence of commands defined by method **3800** that causes all slaves in set Ej that do not satisfy ln-expression ni to move to set Ek. Step **4233** issues a command that causes all slaves in set Ej to move to set El. Similarly, step **4232** issues a sequence of commands defined by method **3800** that causes all slaves in set Ek that do not satisfy ln-expression ni to move to set Ej. Step **4234** issues a command that causes all slaves in set Ek to move to set El. The iteration ends after the last ln-expression, nP, has been processed by either steps **4231** and **4233**, or steps **4232** and **4234**. That terminates the method.

Method **3100** is recoded in FIG. **43** for sum-of-ln-expressions. Method **4300** takes as input **4301** the same input as do methods **4100** and **4200**. Method **4300** outputs a configuration **4302** whereby all slaves in set Ej that satisfy the sum-of-ln-expressions condition **4301** are moved to either set Ek or set El depending on the number of ln-expressions. If the condition has an even number of ln-expressions, set El will contain the desired slaves; otherwise, set Ek will. That can be represented mathematically as

$$Ej=Ej^*\sim(n1+n2+\ldots+nN)$$

(if N is odd) $Ek=El+Ej^*(n1+n2+\ldots+nN), El=0$

(if N is even) $El=El+Ej^*(n1+n2+\ldots+nN), Ek=0$

The main step **4320** controls the iteration over all ln-expressions of condition **4301**. For each ln-expression ni, where i varies from 1 to N, step **4330** tests whether i is odd or even. If i is odd, steps **4331** and **4333** are executed for ln-expression ni. If i is even, steps **4332** and **4334** are executed for ln-expression ni. Step **4331** issues a sequence of commands defined by method **3700** that causes all slaves in set Ej that satisfy ln-expression ni to move to set Ek. Step **4333** issues a command that causes all slaves in set El to move to set Ek. Similarly, step **4332** issues a sequence of commands defined by method **3700** that causes all slaves in set Ej that satisfy ln-expression ni to move to set El. Step **4334** issues a command that causes all slaves in set Ek to move to set El. The iteration ends after the last ln-expression, nN, is processed. That terminates the method.

Method **3400** is recoded in FIG. **44** for sum-of-ln-expressions. Method **4400** takes as input **4401** the same input as do methods **4100**, **4200** and **4300**. Method **4400** outputs a configuration **4402** whereby all slaves in set Ej that satisfy the sum-of-ln-expressions condition **4401** are moved to set Ej, Ek or El depending on the number of ln-expressions. If the number of ln-expressions modulo three is one, that is, 1, 4, 7, etc., set Ej will end up containing the desired set of slaves. If the number of ln-expressions modulo three is two, that, 2, 5, 8, etc., set Ek will end up containing the desired set of slaves. If the number of ln-expressions modulo three is zero, that is, 3, 6, 9, etc., set El will end up containing the desired set of slaves. That can be represented mathematically as

(if (N mod **3**)=1) $Ej=El+Ej^*(n1+n2+\ldots+nN), El=0$

(if (N mod **3**)=2) $Ek=El+Ej^*(n1+n2+\ldots+nN), Ej=0$

(if (N mod **3**)=0) $El=El+Ej^*(n1+n2+\ldots+nN), Ek=0$

The main step **4410** controls the iteration over all ln-expressions of condition **4401**. For each ln-expression ni, where i varies from 1 to N, steps **4419** and **4420** test whether i mod **3** is one, two or zero. If one, steps **4421** and **4424** are executed for ln-expression ni. If two, steps **4422** and **4425** are executed for ln-expression ni. If zero, steps **4423** and **4426** are executed for ln-expression ni. Step **4421** issues a sequence of commands defined by method **3800** that causes all slaves in set Ej that do not satisfy ln-expression ni to move to set Ek. Step **4424** issues a command that causes all slaves in set El to move to set Ej. Similarly, step **4422** issues a sequence of commands defined by method **3800** that causes all slaves in set Ek that do not satisfy ln-expression ni to move to set El. Step **4425** issues a command that causes all slaves in set Ej to move to set Ek. Similarly, step **4423** issues a sequence of commands defined by method **3800** that causes all slaves in set El that do not satisfy ln-expression ni to move to set Ej. Step **4426** issues a command that causes all slaves in set Ek to move to set El. The iteration ends after the last ln-expression, nN, is processed. That terminates the method.

For example, the condition A*B+A*C+~A*~C, which would require ten commands if handled by any of methods **2500**, **2800**, **3100** or **3400**, can be rewritten as (B+C)*A+~A*~B*~C and handled by any of methods **4100**, **4200**, **4300** and **4400**, in which case only eight commands are necessary. In the particular case of method **4100** the eight commands are T**12**(B), T**12**(C), T**21**(~A), T**23**(**1**), T**12**(~A), T**21**(B), T**21**(C), and T**23**(l).

As evident by the examples and method descriptions, not all possible transitions of the three-state machine need be available. Methods **2500** and **4100** can be executed on any of three-state machines **400**, **500**, **600**, or **700**. Methods **2800** and **4200** can be executed on any of three-state machines **400** or **500**. Methods **3100** and **4300** can be executed on any of three-state machines **400** or **500**. Methods **3400** and **4400** can be executed on any of three-state machines **400**, **500**, **600** and **800**. Therefore for three-state machines **400** and **500**, methods **2500**, **2800**, **3100**, **3400**, **4100**, **4200**, **4300** and **4400** can be used singly or in combination. For three-state machine **600**, methods **2500**, **3400**, **4100** and **4400** can be used singly or in combination. For three-state machine **700**, only methods **2500** and **4100** can be used singly or in combination. For three-state machine **800**, only methods **3400** and **4400** can be used singly or in combination.

Other state machines are possible as long as any of three-state machines **400**, **500**, **600**, **700** or **800** remains a corner- stone of the architecture. More states may be added and different transition combinations can be used, any of which could be realized by those skilled in the art given the disclosure presented herein, and depending upon the particular specifications desired. Moreover concomitant varia-

tions in the methods herein described and the form by which conditions are expressed and input to the methods will immediately become apparent to those skilled in the art. For example, iteration over elements of an ln-expression could be handled through recursion instead. They can utilize the teachings of this disclosure to create efficient operative embodiments of the system and methods described and claimed. These embodiments are also within the contemplation of the inventor.

I claim:

1. A state machine slave comprising:

three or more states, being at least a first state, a second state, and a third state, the slave being in the first state;

a memory with one or more information values;

a receiving unit for receiving one or more commands in a command sequence, each of the commands specifying a "transfer from state", a "transfer to state", and a primitive condition; and

a processing unit that causes the slave to move to the second state being the "transfer to state" if the first state is the same as the "transfer from state" and one or more of the information values satisfies the primitive condition, the slave being moved to the third state by another command in the command sequence only if the slave is in the second state, and the slave, once moved into to the third state, remaining in the third state,

wherein two of the states reverse roles after an end of one or more subsequences in the sequence of commands.

2. A state machine slave, as in claim **1**, wherein said at least two of the three states exchange roles with each other after an end of one or more subsequences in the sequence of commands as follows: the first and second states reverse roles.

3. A state machine slave, as in claim **2**, where the first and second states reverse roles at the end of each subsequence corresponding to a term in a sum of ln-expression.

4. A state machine slave, as in claim **3**, where the sum of ln-expression is a sum of products.

5. A state machine slave, as in claim **1**, wherein said at least two of the three states exchange roles with each other after an end of one or more subsequences in the sequence of commands as follows: the second and third states reverse roles.

6. A state machine slave, as in claim **5**, where the second and third states reverse roles at the end of each subsequence corresponding to a term in a sum of ln-expression.

7. A state machine slave, as in claim **6**, where the sum of ln-expression is a sum of products.

8. A state machine slave, as in claim **1**, wherein said at least two of the three states exchange roles with each other after an end of one or more subsequences in the sequence of commands as follows: the first state assumes the role of the second state, the second state assumes the role of the third state, and the third state assumes the role of the first state.

9. A state machine slave, as in claim **8**, where the first, second, and third states cycle roles at the end of each subsequence corresponding to a term in a sum of ln-expression.

10. A state machine slave, as in claim **9**, where the sum of ln-expression is a sum of products.

11. A system for selecting a subset of slaves that satisfy a selection criterion, comprising:

a master unit for communicating a selection criterion command sequence to a plurality of slaves, the selection criterion command sequence representing the selection criterion;

each slave of said plurality of slaves at any time being in one of three or more different possible selection states, said states being used for processing the selection criterion command sequence and for determining whether or not said each slave satisfies the selection criterion represented by the selection criterion command sequence;

a memory in said each slave containing at least one information value for evaluation of primitive selection conditions;

a receiving unit in said each slave for receiving the selection criterion command sequence from the master unit, each command in the selection criterion command sequence specifying a "from" state, a "to" state, and a primitive selection condition; and

a processing unit in said each slave that processes each successive command in the selection criterion command sequence by moving the slave into the "to" state if and only if the slave is already in the "from" state and the at least one information value satisfies the primitive condition,

whereby said each slave is deemed to be selected if said each slave is in a particular selection state after the processing unit in said each slave has processed the selection criterion command sequence.

12. A system, as in claim **11**, where the master unit is a base station, the slaves are radio frequency tags, and the base station communicates with the radio frequency tags with a radio frequency signal.

13. A system, as in claim **11**, where the selection criterion command sequence moves one or more first sets of slaves from a first state to a second state, then moves one or more second sets of slaves from the second state to the first state, and then moves the slaves remaining in the second state to a third state.

14. A system, as in claim **11**, where the selection criterion command sequence moves one or more first sets of slaves from a first state to a second state, then moves one or more second sets of slaves from the second state to the first state, and then moves the slaves remaining in the first state to a third state.

15. A system, as in claim **11**, where the selection criterion command sequence moves one or more first sets of slaves from a first state to a second state and then moves one or more second sets of slaves from the second state to a third state.

16. A system, as in claim **11**, where the selection criterion command sequence moves one or more first sets of slaves from a first state to a second state and then moves one or more second sets of slaves from the first state to a third state.

17. A method for selecting a subset of slaves that satisfy a selection criterion, comprising the steps of:

storing at least one information value in each of a plurality of slaves for evaluating primitive selection conditions;

representing a selection criterion as a selection criterion command sequence, each command in the selection criterion command sequence specifying a "from"state, a "to" state, and a primitive selection condition;

communicating the selection criterion command sequence to the plurality of slaves, each slave of said plurality of slaves at any time being in one of three or more different selection states used for processing the selection criterion command sequence and for determining whether or not said each slave satisfies the selection criterion represented by the selection criterion command sequence;

processing each successive command in the selection criterion command sequence at said each slave by moving said each slave into the "to" state if and only if the slave is already in the "from" state and the at least one information value satisfies the primitive condition; and then

deeming said each slave in a particular selection state to be selected.

18. A state machine slave for responding to a selection criterion command sequence from a master unit, the selection criterion command sequence implementing a selection criterion, the state machine slave comprising:

first, second and third selection states for processing the selection criterion command sequence and for determining whether or not the slave satisfies the selection criterion represented by the selection criterion command sequence;

a memory containing at least one information value for evaluating primitive selection conditions;

a receiving unit for receiving the selection criterion command sequence, each command in the selection criterion command sequence specifying a "from" state, a "to" state, and a primitive selection condition; and

a processing unit that process each successive command in the selection criterion command sequence by moving the slave into the "to" state if and only if the slave is already in the "from" state and the at least one information value satisfies the primitive condition,

whereby the slave is deemed to be selected if the slave is in a particular one of the three states after the processing unit has processed the selection criterion command sequence.

* * * * *

EXHIBIT C

US006286762B1

(12) **United States Patent**
Reynolds et al.

(10) **Patent No.: US 6,286,762 B1**
(45) **Date of Patent: Sep. 11, 2001**

(54) **METHOD AND APPARATUS TO PERFORM A PREDEFINED SEARCH ON DATA CARRIERS, SUCH AS RFID TAGS**

(75) Inventors: **Andrew E. Reynolds**, Bothell; **Christopher A. Wiklof**, Everett; **Daniel B. Bodnar**, Duvall, all of WA (US)

(73) Assignee: **Intermec IP Corp.**, Beverly Hills, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/401,363**

(22) Filed: **Sep. 21, 1999**

(51) **Int. Cl.**[7] **G06K 7/10**
(52) **U.S. Cl.** **235/472.01**; 235/472.02
(58) **Field of Search** 235/472.01, 472.02

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,801,371 * 9/1998 Kahn et al. 235/472.01
6,027,021 * 2/2000 Kumor 235/472.01
6,097,301 * 8/2000 Tuttle 340/693.9
6,104,333 * 8/2000 Wood, Jr. 341/173
6,127,928 * 10/2000 Isacman 340/572.1

OTHER PUBLICATIONS

Specifying and Installing Amtech Products, Dallas, Texas, Jun. 6–10, 1988, "The AUX–2 Serial Port", pp. 1, 6, and 7.
Command codes for the Amtech Model AI–1200 Reader, Versions 2.1, 2.2 and 2.30, Oct. 11, 1988, pp. 1 and 33.
Amtech Corporation Product Catalog 1194, Readers, 1994, pp. 1–10, 1–11, 1–20, and 1–21.

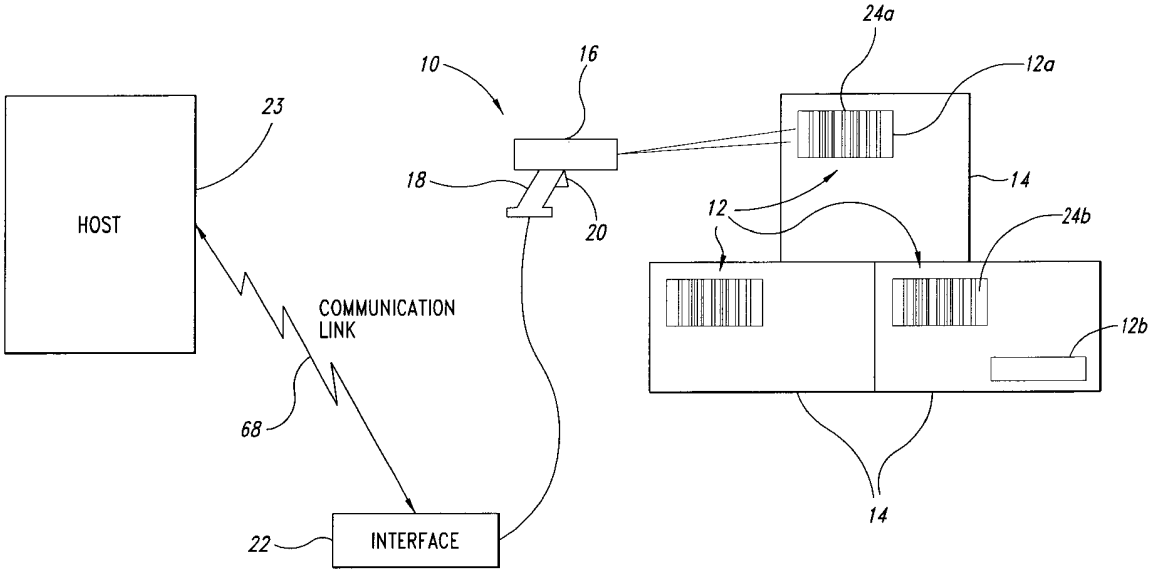
* cited by examiner

Primary Examiner—Harold I. Pitts
(74) *Attorney, Agent, or Firm*—Seed IP Law Group PLLC

(57) **ABSTRACT**

A data carrier reader is capable of executing a number of different reading methods. One method performs an inclusive search, identifying all RFID tags having a characteristic data string that appears on a list of characteristic data strings, for example, stored in a buffer. Another method performs and exclusive search, identifying any RFID tags having a characteristic data string that does not appear on the list. In each method, the data carrier reader provides a consistent and intuitive output the user to identify the successful and unsuccessful operations such as locating a desired RFID tag on the list or missing from the list.

9 Claims, 15 Drawing Sheets



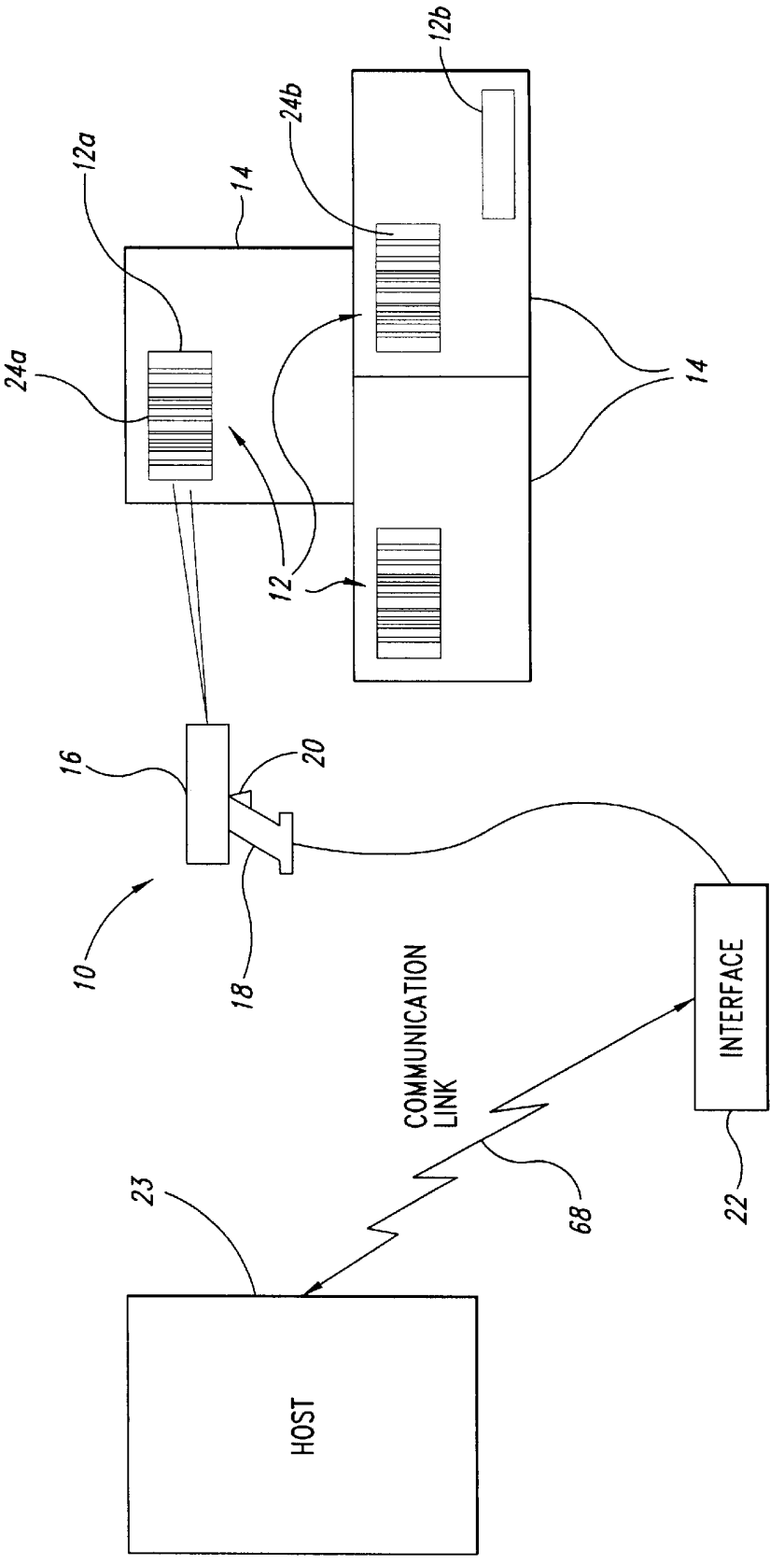

12b
24b
12a
14
24a
14
12
16
20
10
18
COMMUNICATION LINK
INTERFACE
23
68
22
HOST

Fig. 1

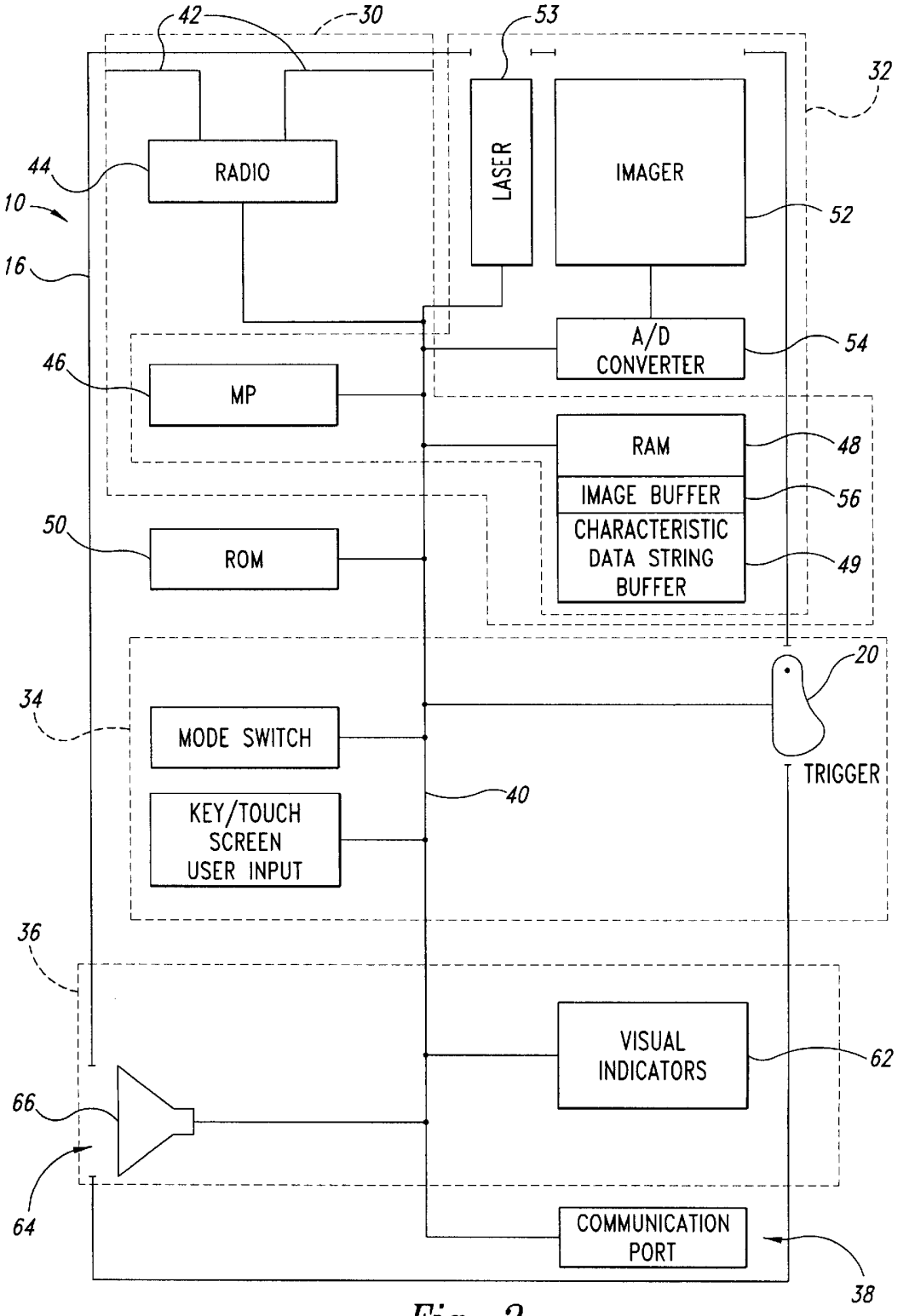

42
30
53
32
44
RADIO
LASER
IMAGER
10
52
16
A/D
CONVERTER
54
46
MP
RAM
48
IMAGE BUFFER
56
50
ROM
CHARACTERISTIC
DATA STRING
BUFFER
49
20
34
MODE SWITCH
TRIGGER
40
KEY/TOUCH
SCREEN
USER INPUT
36
VISUAL
INDICATORS
62
66
64
COMMUNICATION
PORT
38

Fig. 2

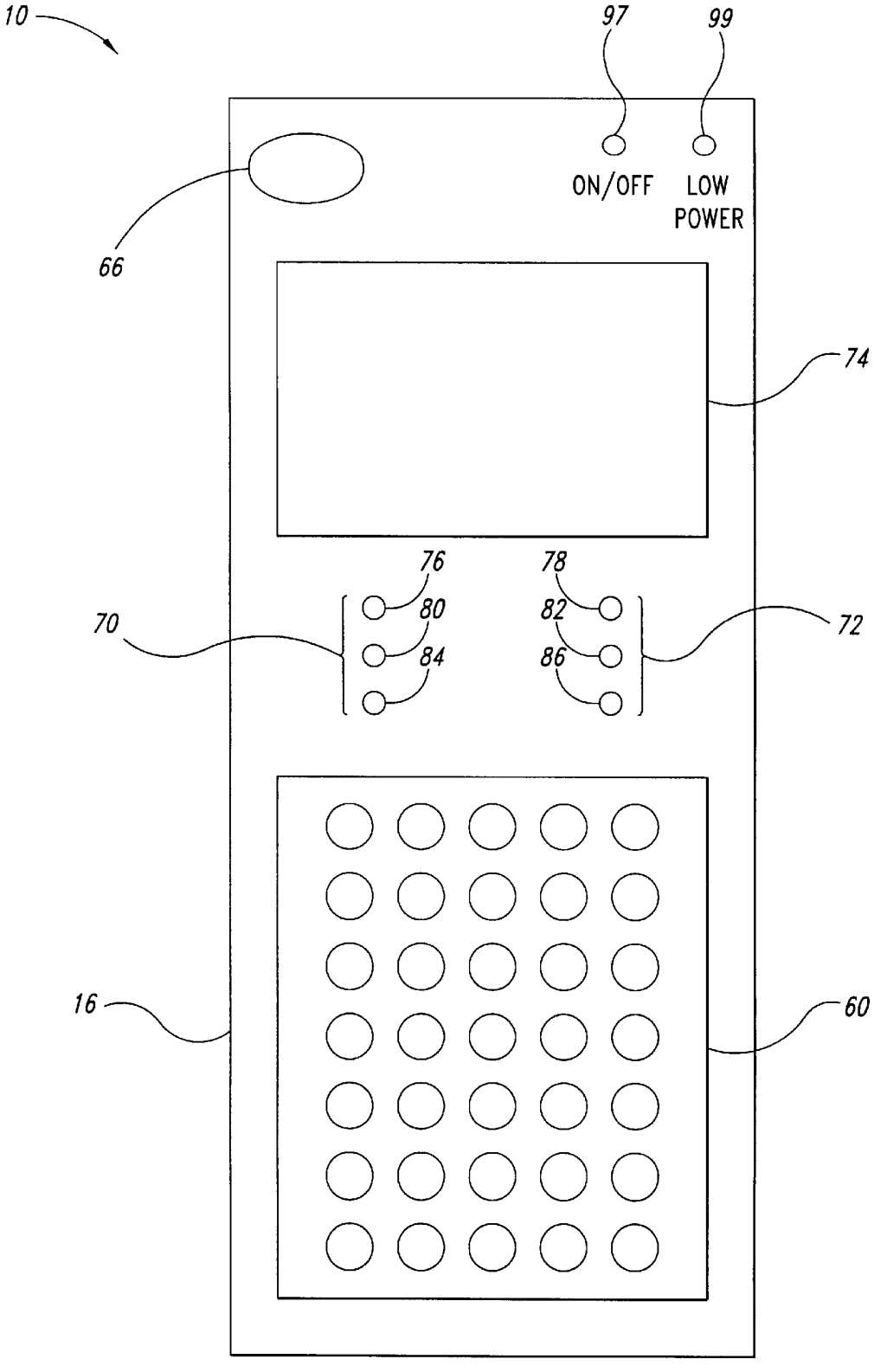


Fig. 3

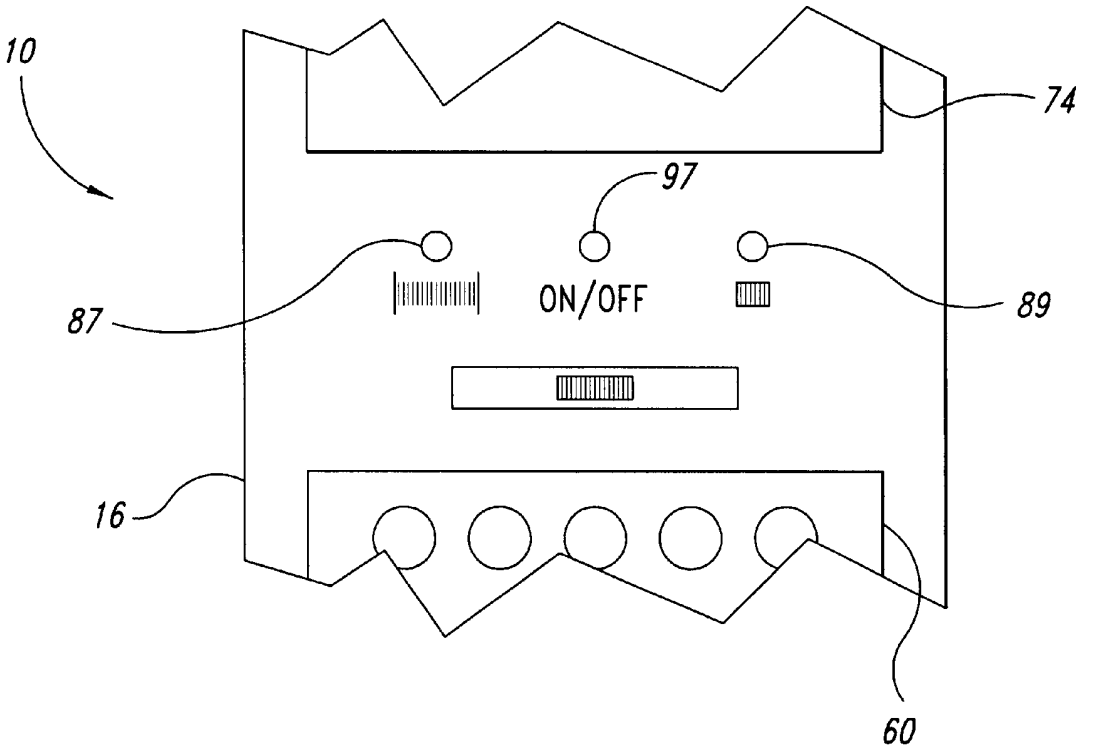
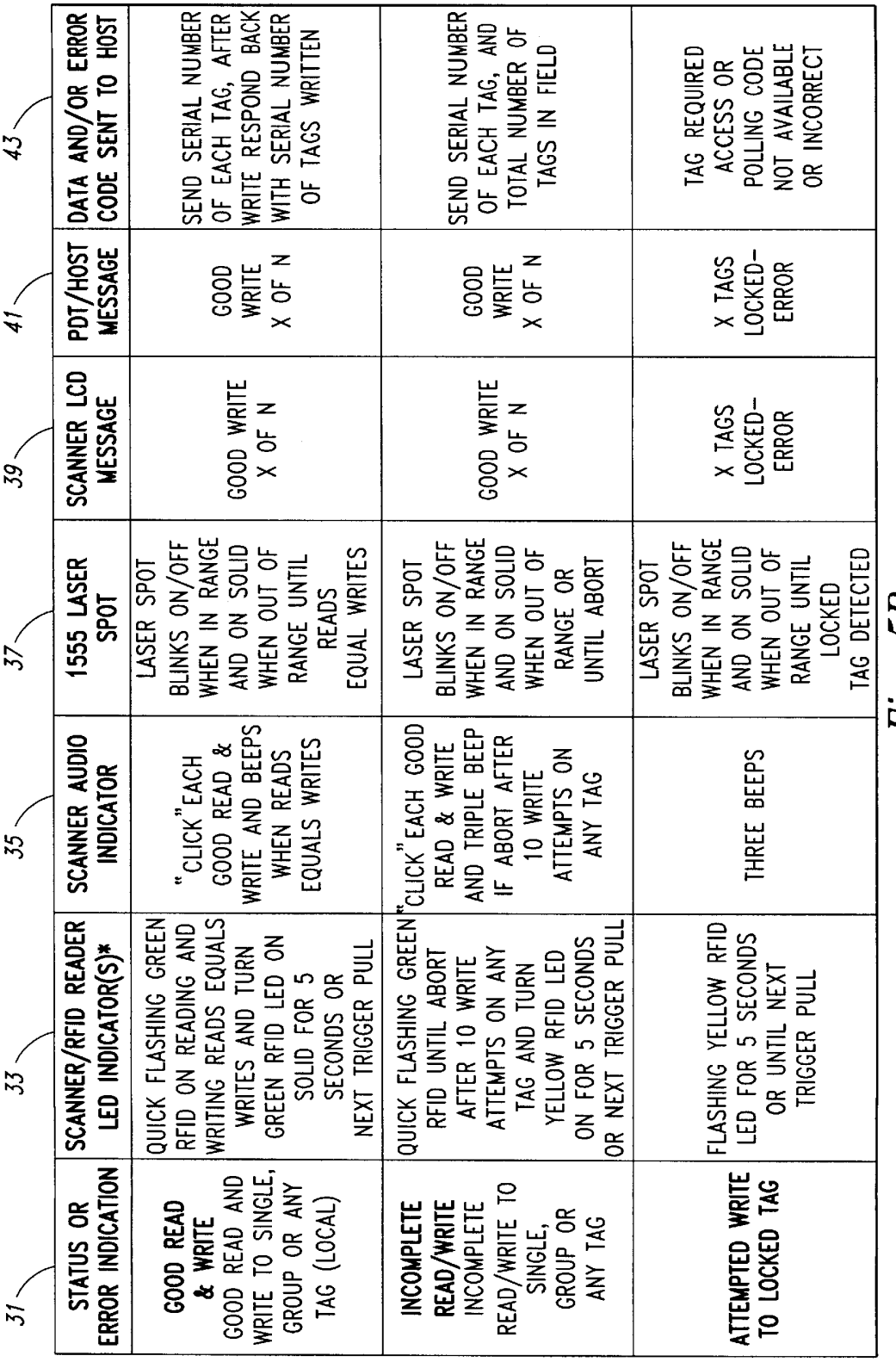


Fig. 4

31 STATUS OR ERROR INDICATION	33 SCANNER/RFID READER LED INDICATOR(S)*	35 SCANNER AUDIO INDICATOR	37 1555 LASER SPOT	39 SCANNER LCD MESSAGE	41 PDT/HOST MESSAGE	43 DATA AND/OR ERROR CODE SENT TO HOST
NO SCANNING OR RFID READING	NONE	NONE	OFF	NONE	NONE	NONE
ATTEMPT RFID READ OR WRITE OF TAGS	YELLOW RFID & POWER LED ON UNTIL SUCCESSFUL READ OR RELEASE OF TRIGGER	NONE	LASER SPOT FLASHING ON/OFF WHEN READING	GOOD READ 0	NONE	NONE
INTERROGATION TOO FAST (UNSUCCESSFUL INTERROGATION)	FLASHING YELLOW RFID LED UNTIL VALID RFID DECODES OR UNTIL TRIGGER IS OFF	CONTINUOUS "CLICKS"	LASER SPOT SOLID ON	"SCANNING TOO FAST"	"SCANNING TOO FAST"	NONE
BAR CODE SCANNING	YELLOW POWER LED ON UNTIL SUCCESSFUL DECODE OR RELEASE OF TRIGGER	NONE	OFF IN DEFAULT BUT CAN BE PROGRAMMED ON AND SET FOR SPECIFIED DURATION	NONE	NONE	NONE
BAR CODE SUCCESSFUL DECODE	GREEN BAR CODE LED ON FOR 5 SECONDS OR NEXT TRIGGER PULL, YELLOW POWER LED OFF	ONE BEEP	OFF	DISPLAY DATA	DISPLAY DATA	DATA

Fig. 5A



31 STATUS OR ERROR INDICATION	33 SCANNER/RFID READER LED INDICATOR(S)*	35 SCANNER AUDIO INDICATOR	37 1555 LASER SPOT	39 SCANNER LCD MESSAGE	41 PDT/HOST MESSAGE	43 DATA AND/OR ERROR CODE SENT TO HOST
GOOD READ & WRITE GOOD READ AND WRITE TO SINGLE, GROUP OR ANY TAG (LOCAL)	QUICK FLASHING GREEN RFID ON READING AND WRITING READS EQUALS WRITES AND TURN GREEN RFID LED ON SOLID FOR 5 SECONDS OR NEXT TRIGGER PULL	"CLICK" EACH GOOD READ & WRITE AND BEEPS WHEN READS EQUALS WRITES	LASER SPOT BLINKS ON/OFF WHEN IN RANGE AND ON SOLID WHEN OUT OF RANGE UNTIL READS EQUAL WRITES	GOOD WRITE X OF N	GOOD WRITE X OF N	SEND SERIAL NUMBER OF EACH TAG, AFTER WRITE RESPOND BACK WITH SERIAL NUMBER OF TAGS WRITTEN
INCOMPLETE READ/WRITE INCOMPLETE READ/WRITE TO SINGLE, GROUP OR ANY TAG	QUICK FLASHING GREEN RFID UNTIL ABORT AFTER 10 WRITE ATTEMPTS ON ANY TAG AND TURN YELLOW RFID LED ON FOR 5 SECONDS OR NEXT TRIGGER PULL	"CLICK" EACH GOOD READ & WRITE AND TRIPLE BEEP IF ABORT AFTER 10 WRITE ATTEMPTS ON ANY TAG	LASER SPOT BLINKS ON/OFF WHEN IN RANGE AND ON SOLID WHEN OUT OF RANGE OR UNTIL ABORT	GOOD WRITE X OF N	GOOD WRITE X OF N	SEND SERIAL NUMBER OF EACH TAG, AND TOTAL NUMBER OF TAGS IN FIELD
ATTEMPTED WRITE TO LOCKED TAG	FLASHING YELLOW RFID LED FOR 5 SECONDS OR UNTIL NEXT TRIGGER PULL	THREE BEEPS	LASER SPOT BLINKS ON/OFF WHEN IN RANGE AND ON SOLID WHEN OUT OF RANGE UNTIL LOCKED TAG DETECTED	X TAGS LOCKED-ERROR	X TAGS LOCKED-ERROR	TAG REQUIRED ACCESS OR POLLING CODE NOT AVAILABLE OR INCORRECT

Fig. 5B

31 STATUS OR ERROR INDICATION	33 SCANNER/RFID READER LED INDICATOR(S)*		35 SCANNER AUDIO INDICATOR	37 1555 LASER SPOT	39 SCANNER LCD MESSAGE	41 PDT/HOST MESSAGE	43 DATA AND/OR ERROR CODE SENT TO HOST
SEARCH FOR MATCHES SEARCH FOR MATCHES WHERE TRIGGER BECOMES ON/OFF TOGGLE SWITCH, i.e., FIRST TRIGGER ACTIVATION TURNS ON AND SECOND TRIGGER ACTIVATION TURNS OFF.	YELLOW LED FLASH ON NON-MATCH. QUICK GREEN FLASHING LED TO EVERY MATCH. HOST PROVIDED SORT TABLE. HOST OR BAR CODE PROGRAMMABLE FOR MATCH OR EXCEPTION.		"CLICK" FOR EACH NON-MATCH AND BEEP FOR EACH MATCH. BEEPER SET TO DEFAULT.	LASER SPOT BLINKS ON/OFF WHEN IN RANGE AND ON SOLID WHEN OUT OF RANGE. FLASHES FASTER WHEN FINDS A MATCH.	"SEARCHING" FLASHING UNTIL "MATCH" ON TOP LINE OF LCD W/ HISTOGRAM SHOWING NUMBER OF NEW READS.	DISPLAY MATCHED DATA.	TRANSFER LIST FROM HOST, TRANSFER MATCHING DATA STRINGS TO HOST.
BUFFER FULL THIS STATUS APPLIES TO AN APPLICATION WHERE THE HAND HELD'S BUFFER BECOMES FULL WITH TAG DATA. FOR EXAMPLE, IN AN INVENTORY OPERATION WHERE A USER CONTINUOUSLY SCANS TAG DATA INTO THE HAND HELD. THE HAND HELD WILL BE UNTETHERED.		YELLOW LED SOLID FOR 5 SECONDS OR UNTIL NEXT USER ENTRY.	TRIPLE BEEP UPON ANY ATTEMPT TO ADD MORE DATA TO THE BUFFER UNTIL IT HAS BEEN UPLOADED OR RESET.	NONE	BUFFER FULL	N/A	TRANSFER DATA FROM BUFFER TO HOST.

Fig. 5C

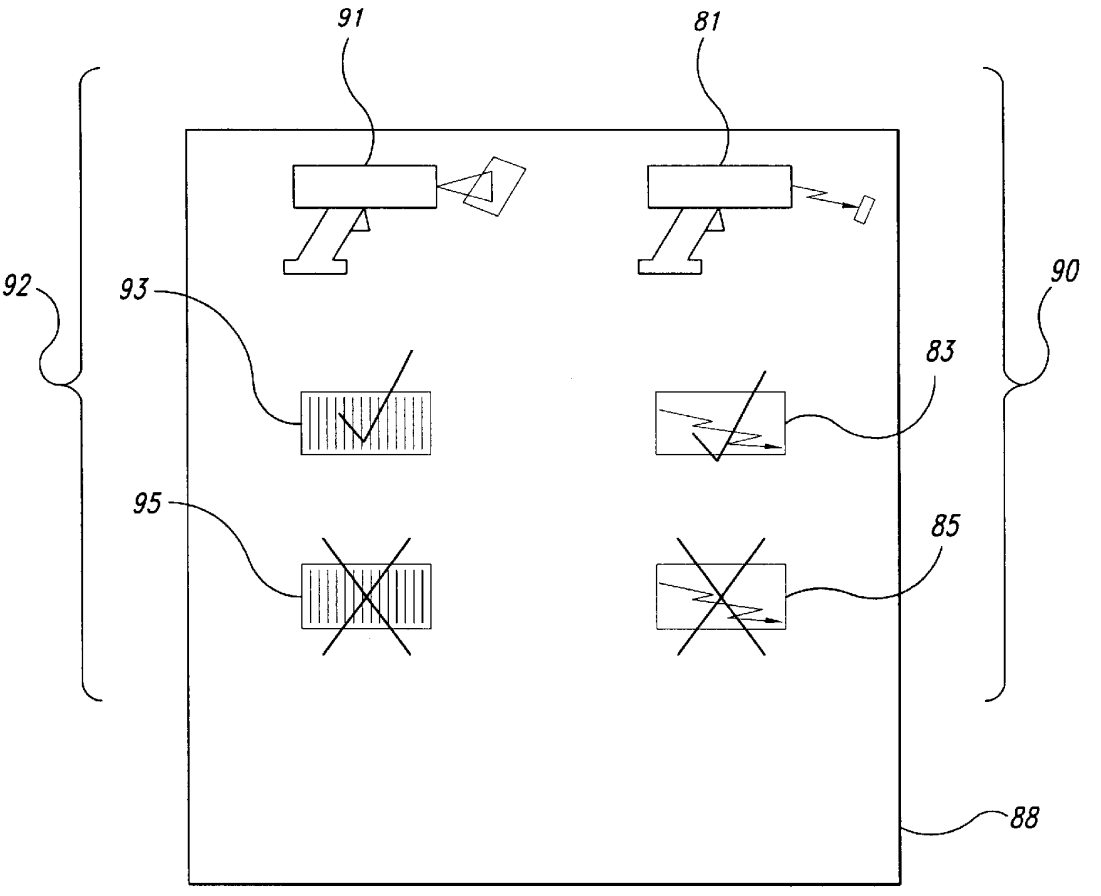

91
81
92
93
90
83
95
85
88

Fig. 6

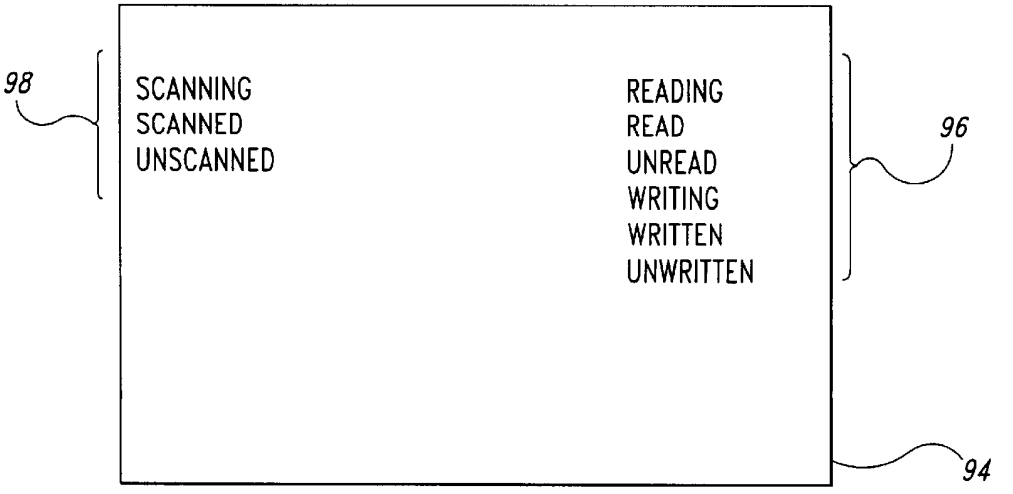

98
SCANNING
SCANNED
UNSCANNED
READING
READ
UNREAD
WRITING
WRITTEN
UNWRITTEN
96
94

Fig. 7

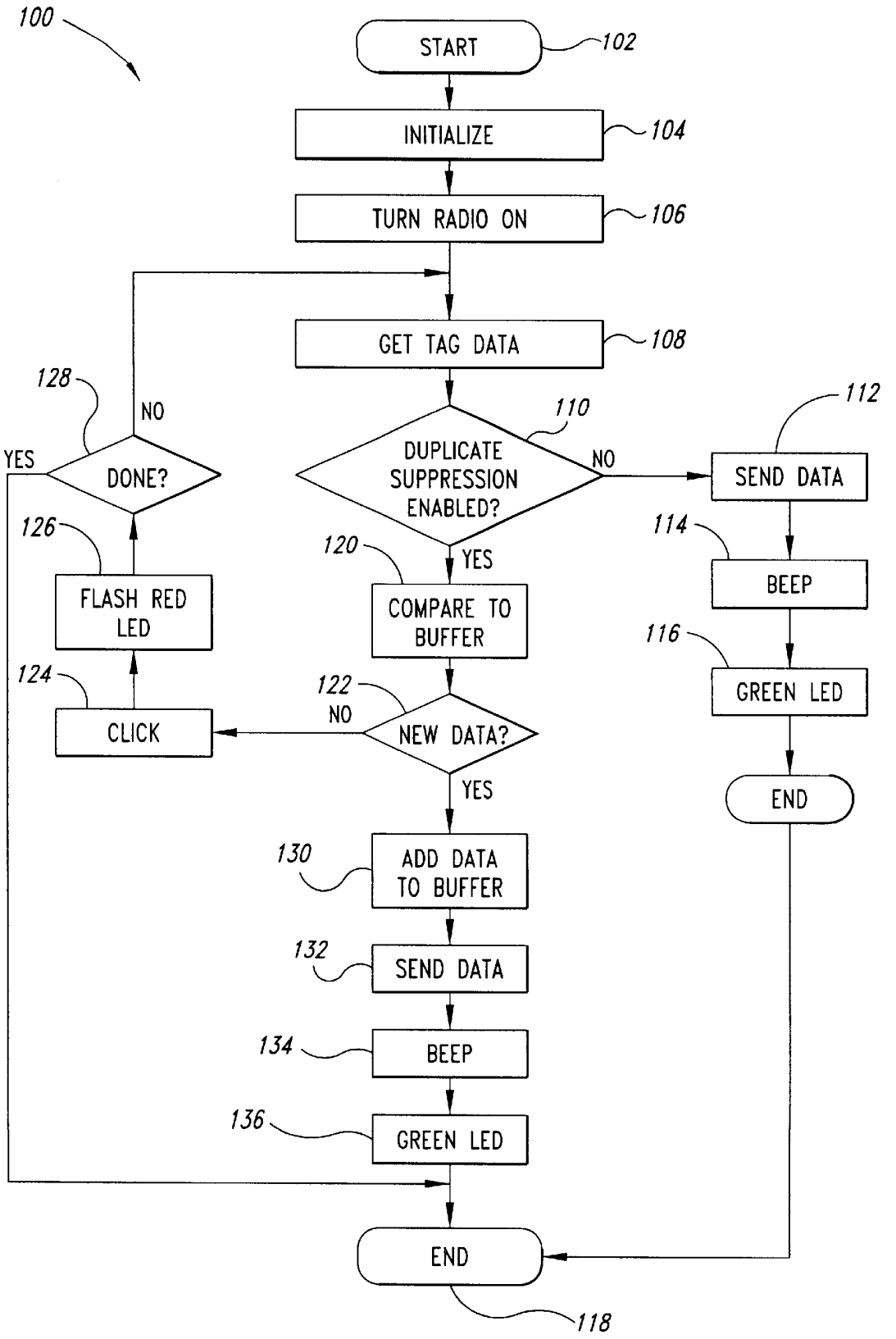

100
START
102
INITIALIZE
104
TURN RADIO ON
106
GET TAG DATA
108
DUPLICATE SUPPRESSION ENABLED?
110
NO
YES
SEND DATA
112
BEEP
114
GREEN LED
116
END
DONE?
128
NO
YES
FLASH RED LED
126
CLICK
124
NEW DATA?
122
NO
YES
COMPARE TO BUFFER
120
ADD DATA TO BUFFER
130
SEND DATA
132
BEEP
134
GREEN LED
136
END
118

Fig. 8

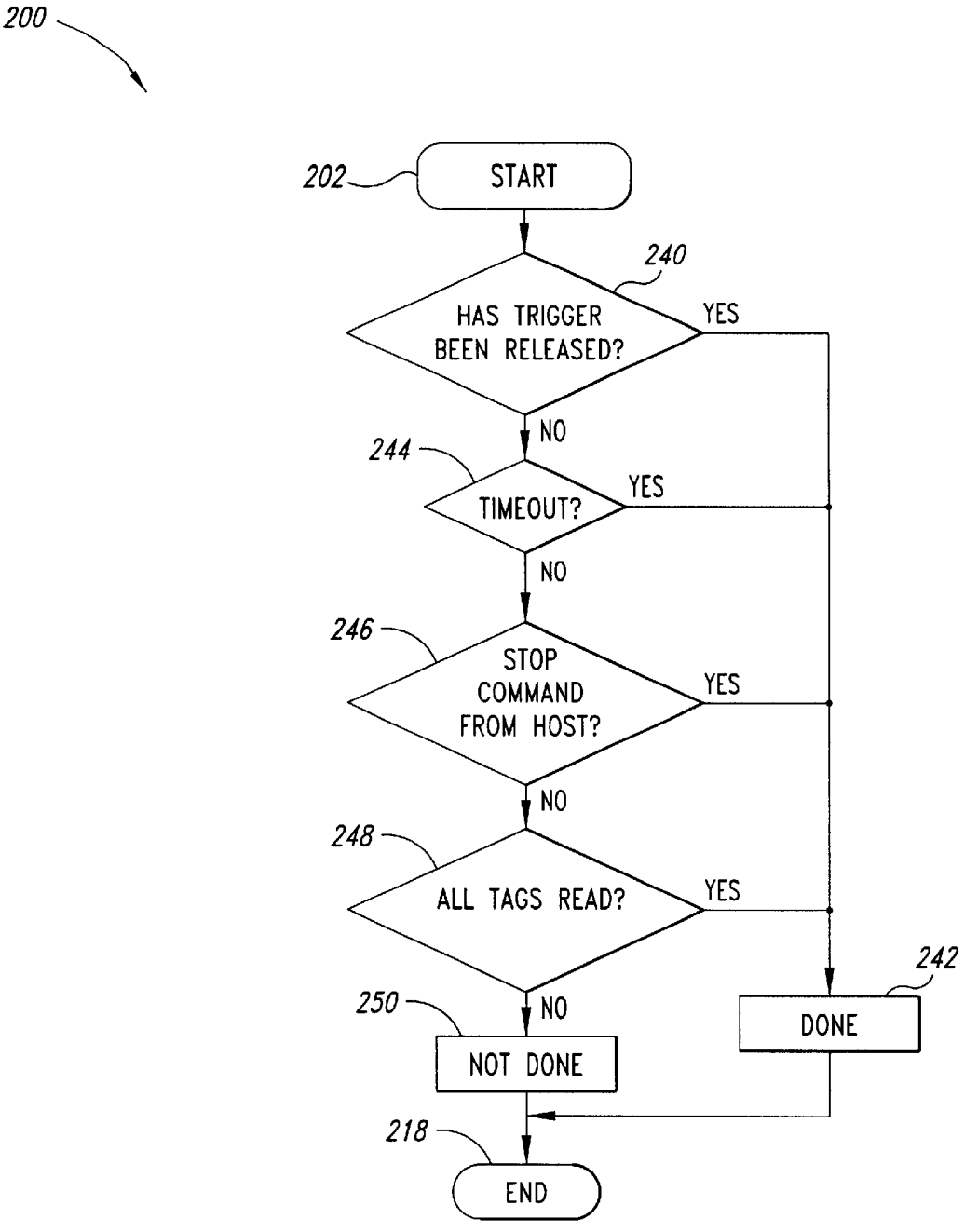

200
202
START
240
HAS TRIGGER BEEN RELEASED?
YES
NO
244
TIMEOUT?
YES
NO
246
STOP COMMAND FROM HOST?
YES
NO
248
ALL TAGS READ?
YES
NO
250
NOT DONE
242
DONE
218
END

Fig. 9

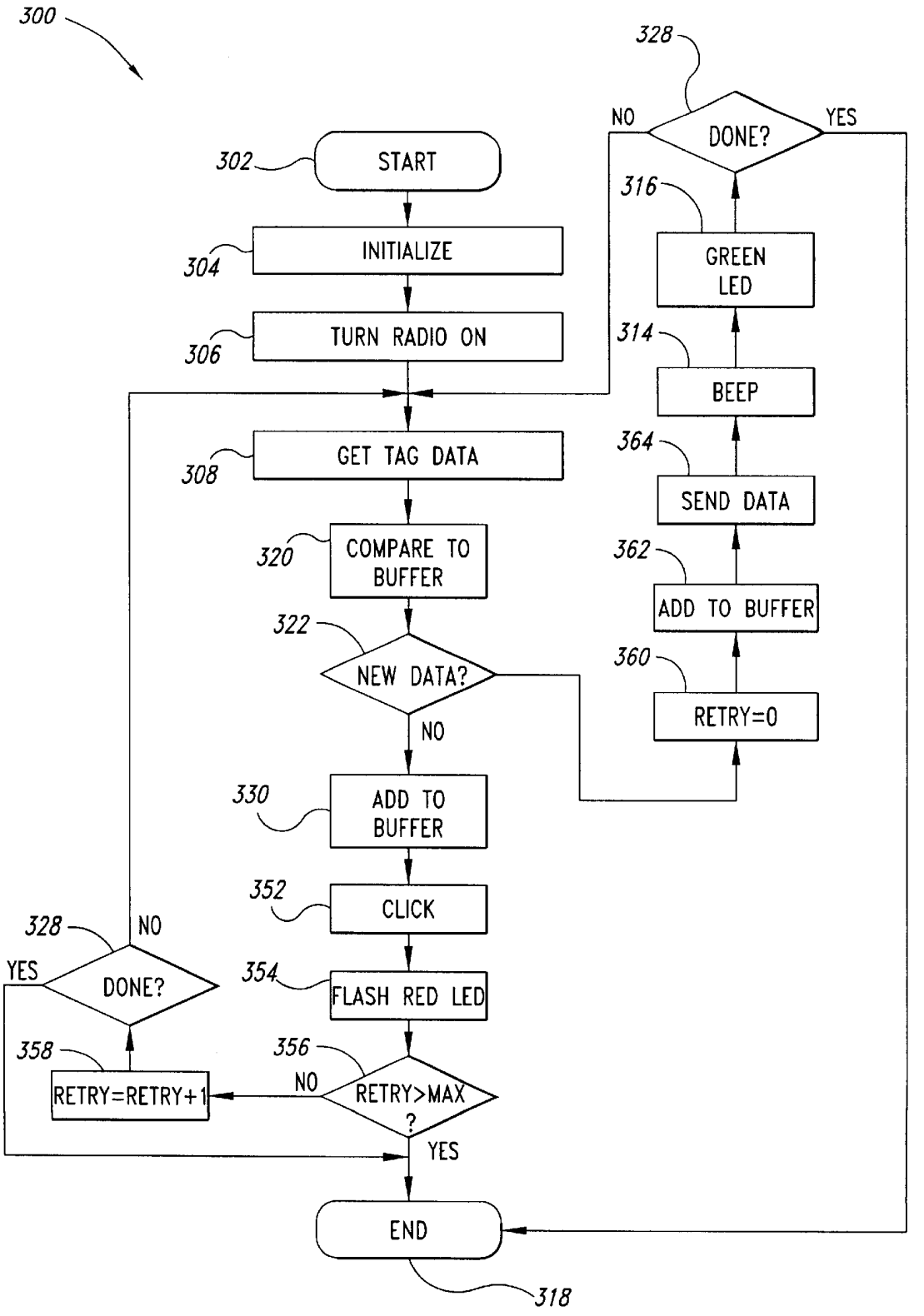

300
START
302
INITIALIZE
304
TURN RADIO ON
306
GET TAG DATA
308
COMPARE TO BUFFER
320
NEW DATA?
322
NO
ADD TO BUFFER
330
CLICK
352
FLASH RED LED
354
RETRY>MAX ?
356
NO
RETRY=RETRY+1
358
YES
DONE?
328
NO
YES
END
318
RETRY=0
360
ADD TO BUFFER
362
SEND DATA
364
BEEP
314
GREEN LED
316
DONE?
328
NO
YES

Fig. 10

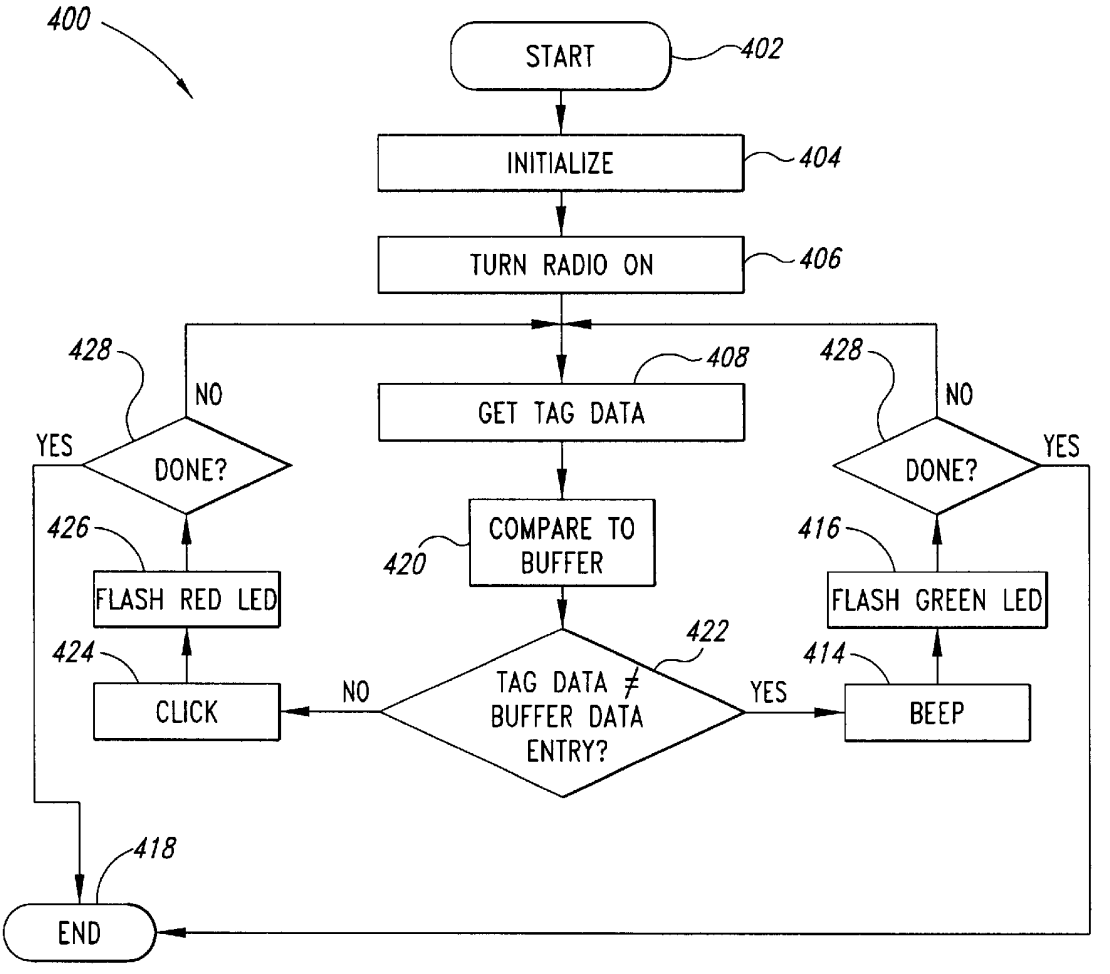

400
START
402
INITIALIZE
404
TURN RADIO ON
406
GET TAG DATA
408
COMPARE TO BUFFER
420
TAG DATA ≠ BUFFER DATA ENTRY?
422
NO
YES
CLICK
424
FLASH RED LED
426
DONE?
428
NO
YES
BEEP
414
FLASH GREEN LED
416
DONE?
428
NO
YES
END
418

Fig. 11

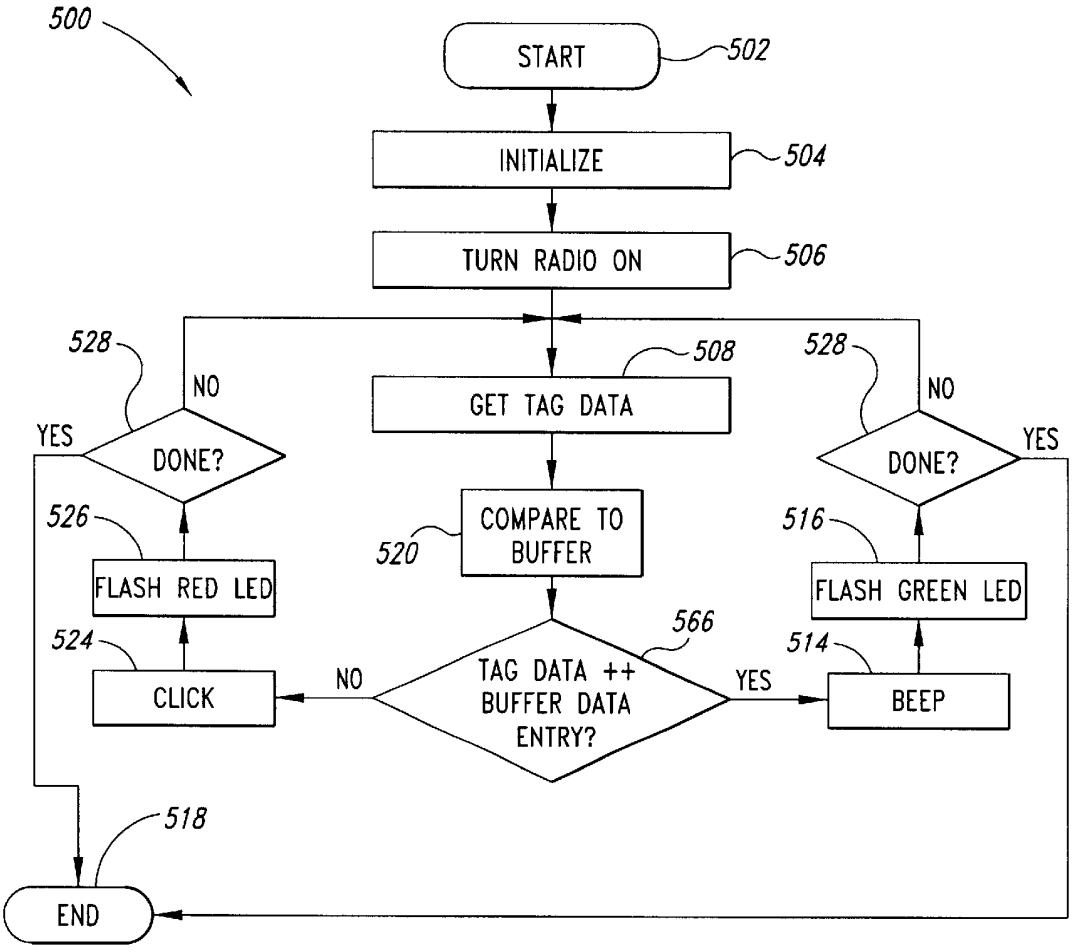

500
START
502
INITIALIZE
504
TURN RADIO ON
506
528
NO
YES
DONE?
508
GET TAG DATA
528
NO
YES
DONE?
526
FLASH RED LED
520
COMPARE TO BUFFER
516
FLASH GREEN LED
524
CLICK
NO
566
TAG DATA ++ BUFFER DATA ENTRY?
YES
514
BEEP
518
END

Fig. 12

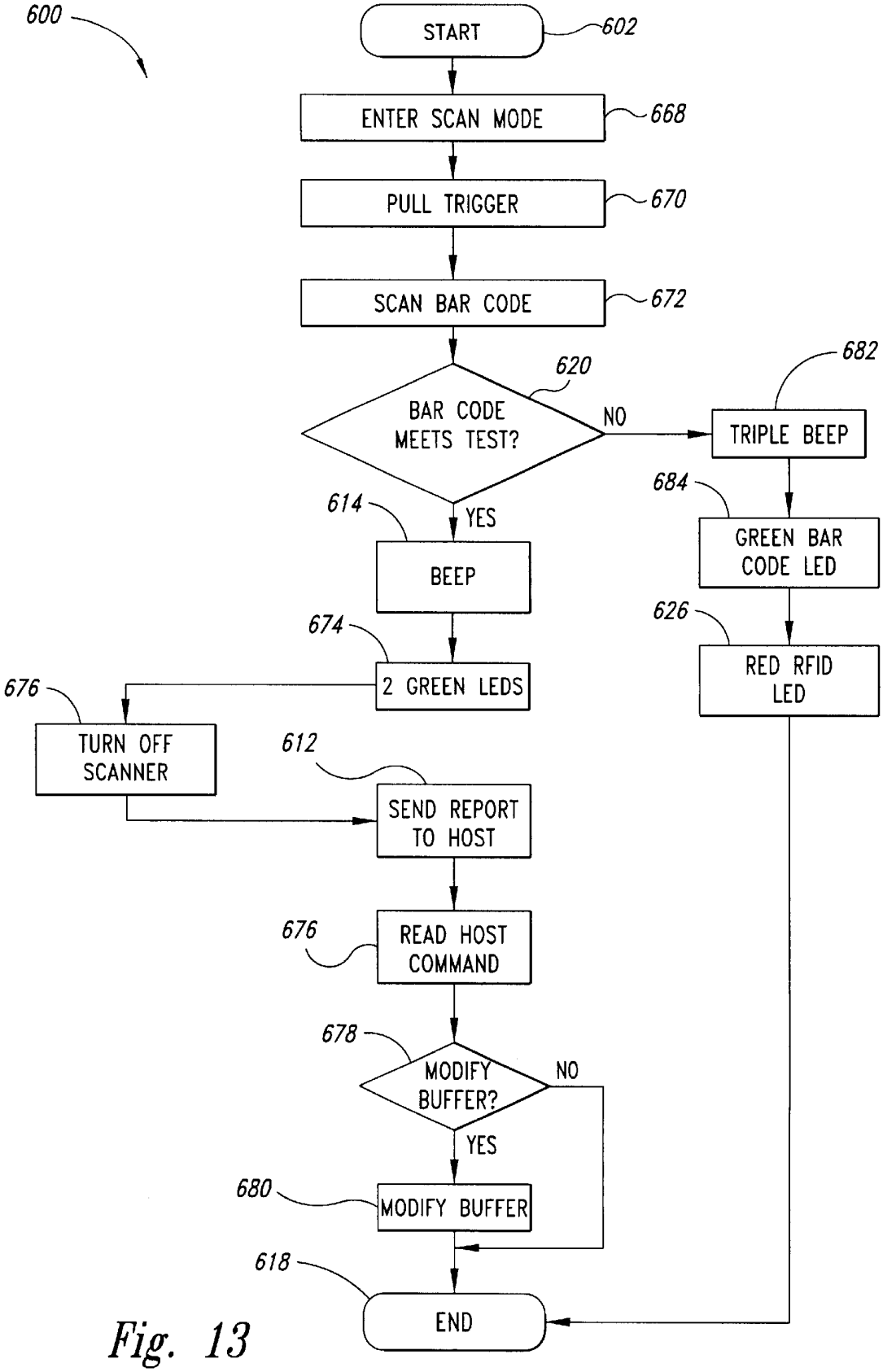

600
START
602
ENTER SCAN MODE
668
PULL TRIGGER
670
SCAN BAR CODE
672
620
BAR CODE MEETS TEST?
NO
682
TRIPLE BEEP
684
GREEN BAR CODE LED
626
RED RFID LED
YES
614
BEEP
674
2 GREEN LEDS
676
TURN OFF SCANNER
612
SEND REPORT TO HOST
676
READ HOST COMMAND
678
MODIFY BUFFER?
NO
YES
680
MODIFY BUFFER
618
END

Fig. 13

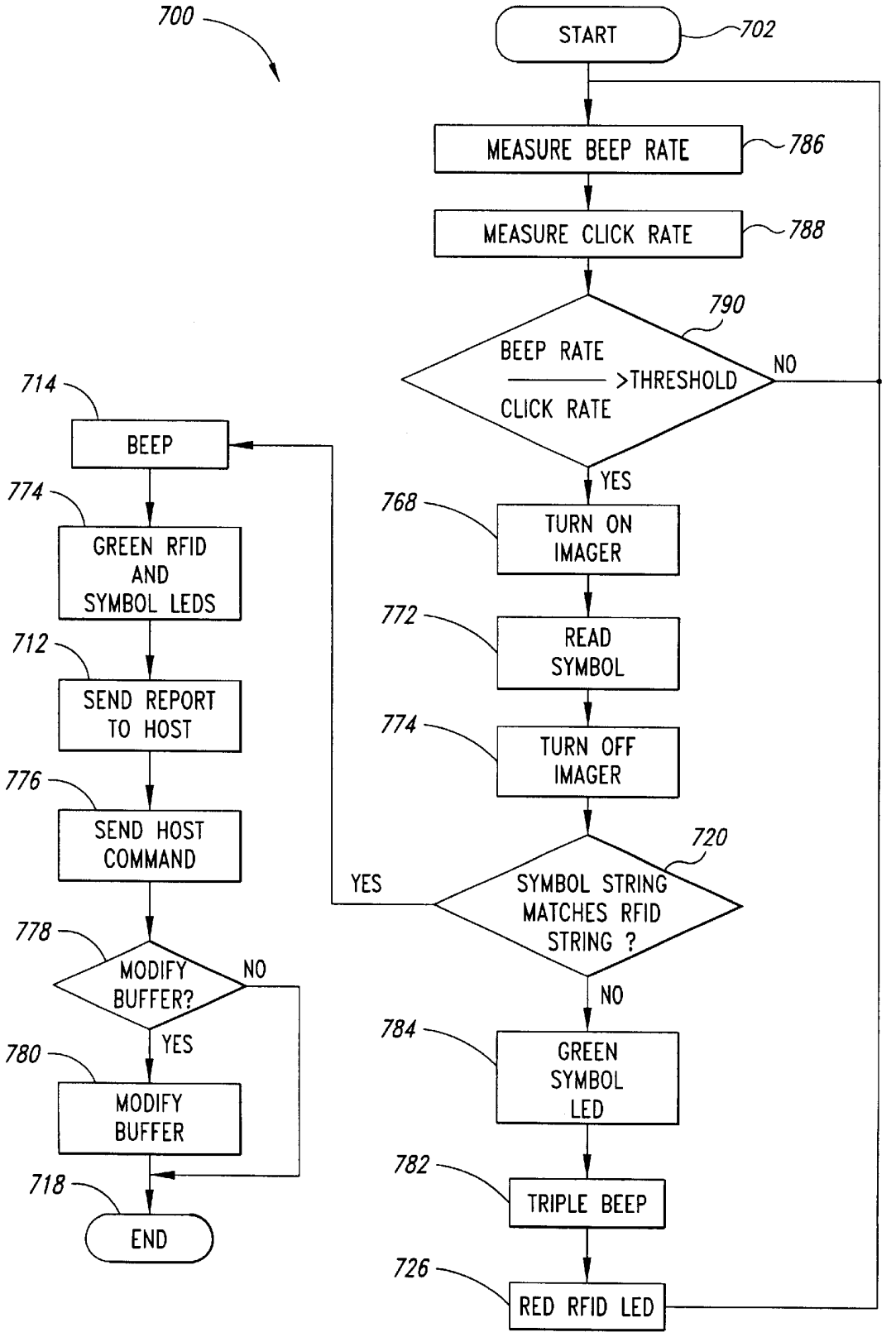

700
START
702
MEASURE BEEP RATE
786
MEASURE CLICK RATE
788
790
BEEP RATE / CLICK RATE >THRESHOLD
NO
YES
768
TURN ON IMAGER
772
READ SYMBOL
774
TURN OFF IMAGER
720
SYMBOL STRING MATCHES RFID STRING ?
YES
NO
784
GREEN SYMBOL LED
782
TRIPLE BEEP
726
RED RFID LED
714
BEEP
774
GREEN RFID AND SYMBOL LEDS
712
SEND REPORT TO HOST
776
SEND HOST COMMAND
778
MODIFY BUFFER?
NO
YES
780
MODIFY BUFFER
718
END

Fig. 14

METHOD AND APPARATUS TO PERFORM A PREDEFINED SEARCH ON DATA CARRIERS, SUCH AS RFID TAGS

TECHNICAL FIELD

This application relates to methods and apparatus for reading data carriers such as machine-readable symbols (e.g., barcode symbols, area and/or matrix code symbols) and wireless memory devices (e.g., RFID tags).

BACKGROUND OF THE INVENTION

A variety of methods exist for tracking and providing information about items. For example, inventory items typically carry printed labels providing information such as serial numbers, price, weight, and size. Some labels include data carriers in the form of machine-readable symbols that can be selected from a variety of machine-readable symbologies, such as bar code, and/or area or matrix code symbologies. The amount of information that the symbols can contain is limited by the space constraints of the label. Updating the information in these machine-readable symbols typically requires the printing of a new label to replace the old label.

Data carriers such as memory devices provide an alternative method for tracking and providing information about items. Memory devices permit the linking of large amounts of data with an object or item. Memory devices typically include a memory and logic in the form of an integrated circuit ("IC") and means for transmitting data to and/or from the device. For example, a radio frequency identification ("RFID") tag typically includes a memory for storing data, an antenna, an RF transmitter, and/or an RF receiver to transmit data, and logic for controlling the various components of the memory device. RFID tags are generally formed on a substrate and can include, for example, analog RF circuits and digital logic and memory circuits. The RFID tags can also include a number of discrete components, such as capacitors, transistors, and diodes.

RFID tags can be passive, active or hybrid devices. Active devices are self-powered, by a battery for example. Passive devices do not contain a discrete power source, but derive their energy from an RF signal used to interrogate the RFID tag. Passive RFID tags usually include an analog circuit that detects and decodes the interrogating RF signal and that provides power from the RF field to a digital circuit in the tag. The digital circuit generally executes all of the data functions of the RFID tag, such as retrieving stored data from memory and causing the analog circuit to modulate the RF signal to transmit the retrieved data. In addition to retrieving and transmitting data previously stored in the memory, the RFID tag can permit new or additional information to be stored in the RFID tag's memory, or can permit the RFID tag to manipulate data or perform some additional functions. RFID tags are available from a number of manufacturers, including Texas Instruments, Dallas, Tex., and Omron of Japan.

Another form of memory device is an optical tag. Optical tags are similar in many respects to RFID tags, but rely on an optical signal to transmit data to and/or from the tag.

Additionally, touch memory data carriers are available, for example touch memory devices from Dallas Semiconductor of Dallas, Tex. Touch memory devices are similar to RFID tags but require physical contact with to store and retrieve data.

A user typically secures a data carrier to an item, such as a good, product, or container by way of a pressure sensitive adhesive. The data carrier often encodes information specifically relating to the item such as identifying or destination information. An individual, such as a checkout or inventory clerk, can retrieve data about any given item, for example, by scanning the machine-readable symbol or interrogating the RF tag, optical tag, or touch memory device. Access to the data can be useful at the point of sale, during inventory, during transportation, or at other points in the manufacture, distribution, sale, or use of the tagged item.

Relatively high cost is one of the drawbacks of memory devices, thus, many applications rely on the less expensive printed machine-readable symbols. Another significant drawback is the difficulty of identifying a particular memory device from a group of memory devices. It is particularly difficult to associate the information read from the RFID tag with a physical item or container. The ability to read data from different types of data carriers, for example machine-readable symbols and RFID tags, and/or to associate and manipulate such data can provide numerous benefits in the automatic data collection ("ADC") industry.

SUMMARY OF THE INVENTION

In one aspect a data carrier reader includes an RFID tag reading section and a machine-readable symbol reading section, which can contain some common components. The reader is operable in an RFID tag reading mode and/or a symbol reading mode. The reader provides a consistent and intuitive user interface within, and between, the operating modes. The user interface can include visual, aural and tactile indicators. The visual indicators can include a pattern displayed by indicators on the reader, or projected onto or near the data carrier.

In another aspect, a data carrier reader is capable of executing a number of different reading methods. A method for reading single RFID tags can store read data to a buffer for eventual transmission to a host, and can suppress redundant data. Another method identifies all RFID tags having a characteristic data string that appears on a list. In contrast, another method identifies any RFID tags having a characteristic data string that does not appear on the list. Still another method associates data read from an RFID tag with a particular object or item using a data coded in a machine-readable symbol. In a further method, the machine-readable symbol is automatically read when the RFID tag is within a predetermined proximity of the reader. In each method, a consistent and intuitive output can be provided to the user to identify the successful and unsuccessful operations such as reading an RFID tag or machine-readable symbol.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings, identical reference numbers identify similar elements or acts. The sizes and relative positions of elements in the drawings are not necessarily drawn to scale. For example, various elements may be arbitrarily enlarged and positioned to improve drawing legibility.

FIG. **1** is a partial block diagram, partial front elevational view of a facility including a data carrier reader reading data carriers carried by a number of items, the reader communicate with a host through an interface.

FIG. **2** is a functional block diagram of the reader according to one embodiment of the invention.

FIG. **3** is a top plan view of the reader of FIG. **2**.

FIG. **4** is a partial top plan view of an alternative set of visual indicators for the reader of FIG. **2**.

FIGS. **5A–5C** together form a chart of selected input and output signals for operating the reader of FIG. **2** and the visual indicators of FIG. **4**.

FIG. **6** is a top plan view of a graphic display of the reader of FIG. **3**.

FIG. **7** is a top plan view of an alpha-numeric display of the reader of FIG. **3**.

FIG. **8** is a flowchart showing a method of reading single RFID tags.

FIG. **9** is a flowchart showing a method of determining when a reader is finished reading RFID tags.

FIG. **10** is a flowchart showing a method of reading multiple RFID tags.

FIG. **11** is a flowchart showing a method of performing an inclusive search of RFID tags.

FIG. **12** is a flowchart showing a method of performing an exclusive search of RFID tags.

FIG. **13** is a flowchart showing a method of associating data from an RFID tag with an item using a machine-readable symbol.

FIG. **14** is a flowchart showing a method of automatically imaging a machine-readable symbol based on proximity to an RFID tag to associate data from an RFID tag with an item using the machine-readable symbol.

DETAILED DESCRIPTION OF THE INVENTION

In the following description, certain specific details are set forth in order to provide a thorough understanding of various embodiments of the invention. However, one skilled in the art will understand that the invention may be practiced without these details. In other instances, well-known structures associated with RFID tags, RFID tag readers, one- and two-dimensional symbologies, symbol readers, microprocessors and communication networks have not been shown or described in detail to avoid unnecessarily obscuring descriptions of the embodiments of the invention.

The headings provided herein are for convenience only and do not interpret the scope or meaning of the claimed invention.

Data Carrier Reader

FIG. **1** shows a data carrier reader **10** reading one or more of a number of data carriers, such as the RFID tags **12** on the containers or items **14**. The reader **10** includes a head **16**, a handle **18** and a trigger **20**. An interface **22** can couple the reader **10** to a host **23**, such as a centralized computer, as described in detail below.

The tags **12** can take the form of an RFID tag **12**A that carries a machine-readable symbol **24**A on a visible surface of the tag. Alternatively, the tags **12** can take the form of a separate RFID tag **12**B and machine-readable symbol **24**B. The separate RFID tag **12**B and machine-readable symbol **24**B can be physically associated, for example, securing each to the same physical object, such as the item **14**. The RFID tag **12**A, **12**B and machine-readable symbol **24**A, **24**B can contain logically associated information, for example information related to the item **14** to which the tags **12** are secured, such as identifying and/or shipping information.

As shown in FIG. **2**, the reader **10** contains an RFID tag reading section **30**, a symbol reading section **32**, a user input section **34**, a user output section **36**, and a communications section **38** all coupled by a bus **40**. The bus **40** provides data, commands and/or power to the various sections **30**–**38**. The reader **10** can include an internal power source such as a rechargeable battery (not shown) or can receive power from an external power source such as a wall outlet by way of an electrical cord (not shown). Each of these sections **30**–**38** will be described individually below, although in the illustrated embodiment some of these sections share common components.

RFID Tag Reading Section

FIG. **2** shows the RFID tag reading section **30** of the data carrier reader **10** including an antenna **42** coupled to a radio **44**. The radio **44** is coupled via the bus **40** to a microprocessor **46** and a random access memory ("RAM") **48**. The RAM **48** can include a characteristic data string buffer **49** to temporarily store characteristic data strings, as will be explained in detail below. Alternatively, the reader **10** can include a discrete characteristic data string buffer (not shown). While FIG. **2** shows a single microprocessor **46**, the data carrier reader **10** may include separate dedicated processors for each of the RFID tag and symbol reading sections **30**, **32**.

While a dipole antenna **42** is shown, the data carrier reader **10** can employ other antenna designs. Of course, the antenna can be selected to achieve a particular focus, for example, a highly directional antenna can enhance the ability of the reader **10** to select a single RFID tag **12** out of a group of RFID tags. The radio **44** can take the form of a transceiver capable of transmitting and receiving at one or more of the frequencies commonly associated with RFID tags **12** (e.g., 350 kilohertz, 400 kilohertz, 900 kilohertz). While these frequencies typically fall within the radio frequency range of the electromagnetic spectrum, the radio **44** can successfully employ frequencies in other portions of the spectrum. Antenna design and radios are generally discussed in *The ARRL Handbook for Radio Amateurs*, 76th Ed., American Radio Relay League, Newington, Conn., U.S.A. (1999) (ISBN: 0-87259-181-6), and commonly assigned patent application U.S. Ser. No. 09/280,287, filed Mar. 29, 1999, entitled ANTENNA STRUCTURES FOR WIRELESS COMMUNICATIONS DEVICE, SUCH AS RFID TAG (Atty. Docket No. 480062.648).

A read only memory ("ROM") **50** stores instructions for execution by the microprocessor **46** to operate the radio **44**. As used in this herein, ROM includes any non-volatile memory, including erasable memories such as EEPROMs. The programmed microprocessor **46** can control the radio **44** to emit an interrogation signal, including any required polling codes or encryption, and to receive a return signal from an RFID tag **12**A, **12**B. The programmed microprocessor **46**, RAM **48**, radio **44** and antenna **42** thus form the RFID reading section **30**.

Symbol Reading Section

FIG. **2** also shows the symbol reading section **32** of the data carrier reader **10** including an image sensor **52** and an illumination source, such as the laser **53**. The image sensor **52** can take the form of a one- or two-dimensional charge coupled device ("CCD") array. Alternatively, the reader **10** can employ other known imaging devices, for example laser scanners or Vidicons. In certain embodiments, the data carrier reader **10** can omit the illumination source, for example where the image sensor **52** is a two-dimensional CCD array operable with ambient light. Alternatively, the data carrier reader **10** can rely on other illumination sources, such as light emitting diodes ("LEDs") or a strobe light, that can be positioned to illuminate a desired one of the machine-readable symbols **24**A, **24**B. The reader **10** can employ suitable optics such as lens and mirrors (not shown) for directing light reflected from the machine-readable symbol **24**A, **24**B to the image sensor **52**.

The reader **10** includes an analog-to-digital ("A/D") converter **54**, to transform the analog electrical signals from the image sensor **52** into digital signals for use by the microprocessor **46**. The bus **40** couples the image data from the A/D converter **54** to the microprocessor **46** and the RAM **48**. A portion of the RAM **48** can form an image buffer **56** for

temporarily storing data, such as a captured image data from the image sensor **52**. The ROM **50** contains instructions for the microprocessor **46**, that permit the microprocessor **46** to control the image sensor **52** to capture image data and to decode and/or manipulate the captured image data. The programmed microprocessor **46**, RAM **48**, image sensor **52**, and A/D converter **54**, thus form the symbol reading section **32**.

Symbol reading and decoding technology is well-known in the art and will not be discussed in further detail. Many alternatives for image sensors, symbol decoders, and optical elements that can be used in the reader **10** are taught in the book, *The Bar Code Book*, Third Edition, by Roger C. Palmer, Helmers Publishing, Inc., Peterborough, N.H., U.S.A. (1995) (ISBN 0-911261-09-5).

Communications Section

The communications section **38** includes a communications buffer **47** and a communications port **49**. The communications buffer **47** can temporarily store incoming and outgoing data and/or commands where the communications speed of the reader **10** does not match the communications speed of some external device, such as the interface **22** (FIG. **1**). The communications port **49** provides communications between the reader and external devices. While shown as a hardwire connection to the interface **22** (FIG. **1**), the communications port can be a wireless interface, and can even employ the antenna **42** and radio **44** of the RFID tag reading section **30**. Additionally, the reader **10** can include the interface **22** as an integral part of the reader **10**.

The interface **22** (FIG. **1**) can provide communications over a communications network **68** to the host **23**, allowing transmissions of data and/or commands between the reader **10** and the host **23**. The communications network **68** can take the form of a wired network, for example a local area network ("LAN") (e.g., Ethernet, Token Ring), a wide area network ("WAN"), the Internet, or the World Wide Web ("WWW"). Alternatively or additionally, the communications network **68** can be a wireless network, for example, employing infrared ("IR"), satellite, and/or radio frequency ("RF") communications.

The host **23** can receive from each of a number of the readers **10**, data collected from the RFID tags **12** and machine-readable symbols **24**. The host **23** can use the data with a database, and can automatically manipulate the data, for example to automatically performing inventory or to track shipments.

The host **23** can provide data and commands to each of a number of the readers **10**. For example, the host can share data between the readers **10**, such as providing a list of either located or missing identifiers, as will be discussed in more detail below in reference to inclusive and exclusive searches. The host **23** can provide a command to toggle the reader **10** between an RFID tag reading mode and a symbol reading mode, which is described below in further detail. Thus, the host **23** can command, coordinate and share data between a number of readers **10**. Commonly assigned patent application U.S. patent application Ser. No. 09/401,066, filed Sep. 22, 1999, entitled, "SYSTEM AND METHOD FOR AUTOMATICALLY CONTROLLING OR CONFIGURING A DEVICE, SUCH AS AN RFID READER" (Atty. Docket No. 480062.672) contains teachings that can be used to automatically control or configure the reader **10**.

User Input Section

The user input section **34** includes the trigger **20**, the mode switch **34**, and can include a user input device **58**. The bus **40** couples the mode switch **34** to the microprocessor **46**. In response to selection of the mode switch **34**, the microprocessor **46** switches between the symbol reading mode and the RFID tag reading mode, for example by toggling between the two operating modes. The reader **10** can employ additional operating modes, or switching positions as desired, for example a switch position that places the reader **10** in an OFF state or a WAIT state to conserve energy.

In the symbol reading mode, the microprocessor **46** operates the image sensor **52** to image one of the machine-readable symbols **24**A, **24**B. The microprocessor **46** decodes the imaged symbol to retrieve the data encoded in the machine-readable symbol **24**A, **24**B, such as a respective identifier. In the RFID tag reading mode, the microprocessor **46** operates the radio **44** to emit an interrogation signal and to receive a response from one or more of the RFID tags **12**A, **12**B to the interrogation signal. The microprocessor **46** decodes the response signal to retrieve the data encoded in the RFID tag **12**A, **12**B, such as a respective identifier.

The mode switch **34** can be a membrane switch, mounted to the exterior of the reader **10** for easy selection by the user. The mode switch **34** can additionally, or alternatively, be implemented in the software to supplement or replace the user selectable mode switch on the exterior of the reader **10**. The software implemented switch is particularly useful where the host **23** (FIG. **1**) controls the operating mode of the reader **10**. Alternatively, the mode switch **34** can be implemented as an icon on a touch sensitive display **74**. In further alternatives, the trigger **20** can function as the mode switch **37**. In one instance, the number of successive trigger pulls or activations can determine the operating mode. For example, two successive trigger pulls can select the symbol mode, while three successive trigger pulls selects the RFID mode; or a single trigger pull can cause the reader **10** to read a symbol while a double trigger pull toggles between the symbol and RFID modes. Alternatively, the duration of trigger activation can determine the operating mode. For example, a trigger pull of under 0.5 seconds can select the symbol mode, while a trigger pull of longer than 0.5 seconds can select the RFID mode; or a trigger pull of under 0.5 seconds can cause the reader **10** to read a symbol while a trigger pull of over 0.5 seconds toggles the reader between the symbol and RFID modes. Additionally, or alternatively, the mode switch can be context sensitive, switching modes based on data read from a previously read data carrier **12**A, **12**B, **24**A, **24**B. For example, a previously read RFID tag **12**A can indicate the existence of a symbol **24**A. In response, the data carrier reader **10** can automatically switch into symbol mode and read the symbol **24**A associated with the RFID tag **12**A.

The bus **40** also couples the trigger **20** to the microprocessor **46**. In response to activation of the trigger **20**, the microprocessor **46** can cause the image sensor **52** to image one of the machine-readable symbols **24**A, **24**B when the reader **10** is operating in the symbol reading mode. In at least one embodiment, the microprocessor **46** can also cause the radio **44** and antenna **42** to emit an interrogation signal in response to the activation of the trigger **20** while in the reader **10** is operating in the RFID tag reading mode.

The user input device **58** can take the form of a keypad **60** (FIG. **3**), mouse, touch screen and/or other user operable device to input information and/or commands to the reader **10**. The bus **40** couples the user input device **58** to the microprocessor **46**, to allow the user to enter data and commands.

User Output Section

The user output section **36** includes human-perceptible visual and audio indicators **62**, **64** respectively. The bus **40** couples the visual and audio indicators **62**, **64** to the micro-

processor **46** for control thereby. The visual indicators **62** can take a variety of forms, for example: light emitting diodes ("LEDs"); a graphic display such as a liquid crystal display ("LCD"), and/or an alpha-numeric display such as a 7-segment display. The audio indicator **64** can take the form of one or more dynamic, electrostatic or piezo-electric speakers **66**. The speaker **66** is operable to produce a variety of sounds (e.g., Clicks and Beeps), and/or frequencies (e.g., tones), and to operate at different volumes. The reader **10** can also include tactile indicators such as a vibrating member. The specific operation of the user output section **36** is discussed in more detail below.

FIG. **3** shows a portion of the user interface located on the head **16** of the reader **10**. The user interface includes the elements of the user input section **34**, such as the trigger **20**, the mode switch **34** and the keypad **60**. The user interface also includes the elements of the user output section **36** including the visual indicators **63** and the speaker **66**. In particular, the visual indicators **62** in the illustrated embodiment include a set of RFID related LEDs **70**, a set of machine-readable symbol related LEDs **72**, and a display **74**.

The data carrier reader **10** can additionally, or alternatively, employ the laser **53** as the visual indicator. The laser can be successively pulsed or flashed according to a set of predefined human-recognizable temporal patterns to provide information to the user, such as user indications corresponding to the various reader operations and/or the responses from the date carriers **12**A, **12**B, **24**A, **24**B. Employing the laser **53** as a portion of the user interface provides a number of distinct benefits. For example, operating the laser **53** to provide human-recognizable patterns can eliminate the need for other visual indicators **62**. The data carrier reader **10** can employ multiple illumination sources such as lasers **53** or LEDs of different colors, or an illumination source capable of producing a number of different colors to provide the appropriate user indications, as set out in FIGS. **5**A–**5**C. As discussed in detail below, the human-recognizable patterns can take the form of a predefined sequence of laser flashes of one or more colors, separated by time (i.e., temporal pattern).

The visual and audio indicators **62**, **64** are configured to provide an intuitive user interface consistent across the RFID tag and symbol reading modes. For example, the RFID tag related and symbol related LED sets **70**, **72** each contain green **76**, **78**, yellow **80**, **82** and red **84**, **86** LEDs, in an order or pattern that is consistent between the sets. The particular LED **76**–**86**, as well as the number and/or pattern of flashes, is set such that the same color LEDs flash the same pattern for analogous RF tag reading and symbol reading activities. For example, the yellow LED **80** in the RFID tag related set **70** flashes during the reading of one of the RFID tags **12**A, **12**B (FIG. **1**), while the yellow LED **82** in the machine-readable symbol related set **72** flashes during the reading of one of the machine-readable symbols **24**A, **24**B (FIG. **1**). The reader **10** responds to a successful read of the RFID tag **12**A, **12**B or machine-readable symbol **24**A, **24**B by illuminating the corresponding green LED **76**, **78**, respectively, for a set period of time such as 5 seconds. The red LEDs **84**, **86** can indicate unsuccessful or incomplete operations. The user receives visual feedback, where the color, position and sequence of the visual indicators **62** is consistent within, and across the RFID tag and symbol operating modes. Consistent feedback can reduce training time and costs, and can lead to more efficient operation of the reader **10**.

Similar to the visual indicators **62**, the speaker **66** provides consistent feedback within and across the operating modes. In the illustrated embodiment, the speaker **66** emits a "beep" or a "click" sound, although the speaker **66** can emit different and/or additional sounds. The speaker **66** can emit, for example, a single beep each time either an RFID tag **12**A, **12**B or a machine-readable symbol **24**A, **24**B is successfully read. When searching a field of RFID tags **12**A, **12**B for one or more particular tags, the speaker **66** can emit a click for each non-match and a beep for each match.

The user interface can also include an ON/OFF indicator **97**, and/or a Low Power indicator **99** to identify the operating condition of the reader **10**.

FIG. **4** shows an alternative set of visual indicators for the reader **10**. his alternative embodiment, and those alternative embodiments and other alternatives described herein, are substantially similar to previously described embodiments, and common acts and structures are identified by the same reference numbers. Only significant differences in operation and structure are described in detail below.

The reader **10** of FIG. **4** employs only three LEDs to simplify switching while providing the human-perceptible visual indications. A two state LED serves as the machine-readable symbol related indicator **87**. The machine-readable symbol indicator **87** produces no light in an OFF state and a Green light in an ON state. A three state LED serves as the RFID related indicator **89**. The RFID related indicator **89** produces a Green light in first ON state, a Yellow light in second ON state, and NO light in an OFF state. A two state LED serves as the ON/OFF indicator **97**. The ON/OFF indicator produces a Yellow light, or No light. The ON/OFF indicator is proximate the machine-readable symbol related and RFID related indicators **87**, **89**. In FIG. **4**, the mode switch **34** takes the form of a toggle or slider switch, having a neutral position (center), a symbol mode position (left of center) and an RFID mode position (right of center). The positions are consistent with the corresponding visual indicators **87**, **89**, respectively.

FIGS. **5**A–C describe a variety of input and outputs signals for the reader **10**, and particularly for the audio indicator **64** and laser **53** of FIG. **2**, and for the visual indicators **87**, **89**, **97** of FIG. **4**. While the table is self-explanatory, a brief description of the columns follows. Column **31** defines a reader status or error conditions corresponding to reader activities. Column **33** describes the operation of the visual indicators **87**, **89**, **97** of FIG. **4**, in response to the various reader status or errors conditions. Similarly, column **35** describes the operation of the audio indicator **64** in response to the various reader status or error conditions **33**. Column **37** describes the operation of the laser to produce the desired human-recognizable patterns corresponding to the various reader status or errors conditions **31**. Column **39** describes messages for display on the display **74** corresponding to the various reader status or errors conditions **31**. Column **41** describes PDT/Host messages corresponding to the various reader status or errors conditions **31**. Column **43** describes data and/or error codes sent to the host **33**, corresponding the various reader status or errors conditions **31**. As discussed above, these user indications provide a consistent interface for the user within and across the operating modes, permitting the user to efficiently operate the reader **10**.

The display **74** can additionally, or alternatively, provide the user other visual indications. For example, a graphical display **88** (FIG. **6**), can employ a first set of icons **90** to indicate RFID tag activities and a second set of icons **92** to indicate symbol reading activities. (Note, typically only a single icon will be displayed at a time, although multiple icons are shown in FIG. **6** for the convenience of this

description.) For example, screen icons **81**, **83** and **85** can represent RFID reading, successful reading of the RFID tag **12**A, **12**B, and unsuccessful reading of RFID tag **12**A, **12**B, respectively. Similarly, screen icons **91**, **93** and **95** can represent machine-readable symbol reading, successful reading of the machine-readable symbol **24**A, **24**B, and unsuccessful reading of the machine-readable symbol **24**A, **24**B, respectively.

Similarly, an alpha-numeric display **94** (FIG. **7**) can employ a first set of words **96** to indicate RFID tag activities and a second set of words **98** to indicate symbol reading activities. (Again, typically only a single word will be displayed at a time, although multiple are shown in FIG. **7** for the convenience of this description.) The display **94** is self-explanatory and in the interest of brevity will not be further described. Other visual indications, as well as audio and tactile indications are of course possible.

Selected Methods of Operation

Different methods of operating the reader **10** or a reader having similar capabilities are disclosed below. As set out in the below methods, the intuitive and consistent operation of the user interface within and across operating modes can provide numerous benefits. While several methods are set out for illustration, other methods employing similar techniques are within the scope of the invention. Also, the following descriptions employ certain descriptions of user outputs (e.g., Beep, Click, Red LED, Yellow LED, and Green LED) for convenience of description. Those skilled in the art will appreciate that other sounds, colors, visual, tactile indications, and/or other human-perceptible indications could be used.

Single Tag Read Mode

FIG. **8** shows a method **100** of reading RFID tags **12**A–**12**B (FIG. **1**) employing the reader **10** (FIGS. **1**–**3**). Turning on the reader **10**, or switching into the RFID tag reading mode, can automatically cause the microprocessor **46** to start the method **100** in step **102**. Alternatively, or additionally, the user can cause the microprocessor **46** to start the RFID tag reading method **100** by selecting an appropriate key from the keypad **60** or icon from the display **74**. Upon starting in step **102**, the microprocessor **46** can perform an initialization process, for example loading appropriate operating instructions from the ROM **50** to the RAM **48**, initializing the characteristic data string buffer **49** and/or performing a series of systems checks on the various component and subsystems of the reader **10**, as set out in step **104**.

Under the instructions loaded in the RAM **48**, the microprocessor **46** activates the radio **44** in step **106**. In step **108**, the radio **44** receives data from the RFID tags **12**A, **12**B. The radio **44** can emit an interrogation signal to cause the RFID tags **12**A, **12**B to respond, or, the radio **44** can simply receive signals from RFID tags **12**A, **12**B that emit signals without interrogating the RFID tags. A variety of passive, active and hybrid RFID tags **12**A, **12**B are known in the art and will not be discussed in further detail. A discussion of RFID tags can be found in commonly assigned patent applications: U.S. Ser. No. 09/173,539, filed Oct. 15, 1998, entitled WIRELESS MEMORY DEVICE AND METHOD OF MANUFACTURE (Atty. Docket No. 480062.630); U.S. Ser. No. 09/164,203, filed Sept. 30, 1998, entitled MEMORY TAG AND METHOD OF MANUFACTURE (Atty. Docket No. 480062.632); U.S. Ser. No. 09/173,137, filed Oct. 15, 1998, entitled RF TAG HAVING STRAIN RELIEVED STIFF SUBSTRATE AND HYDROSTATIC PROTECTION FOR A CHIP MOUNTED THERETO (Atty. Docket No. 480062.635); and U.S. Ser. No. 09/164,200, filed Sept. 30, 1998, entitled CHIP PLACEMENT ON SMART LABELS (Atty. Docket No. 480062.642).

In step **110**, the microprocessor **46** determines whether duplicate tag data should be suppressed. If suppressed, previously read or acquired data will not be stored or reported a second time. Suppression can be a user selection, or can be a selection transferred from the host **23**, or can be preset, for example by the reader manufacturer or owner. If suppression is not active, the reader **10**, in step **112**, automatically transmits the read data, for example to the host **23**, and provides an indication to the user that the data has been received and transmitted. To provide the indication, the reader **10** activates the speaker **66** to emit a single "beep" and activates the Green RFID related LED **76** for a short time, in steps **114**, **116**, respectively. Control passes to an end of the routine **100**, in step **118**.

If suppression is active, the microprocessor **46**, compares a characteristic data string from the received data to other characteristic data strings stored in the characteristic data string buffer **49**, in step **120**. The characteristic data string can be any string of characters stored in the RFID tags **12**A, **12**B that permit the reader **10** to determine whether a particular RFID tag **12**A, **12**B has been read more than once. For example, the characteristic data string can be a unique identifier programmed into each of the RFID tags **12**A, **12**B. Alternatively, the characteristic data string can be the entire set of data stored in the RFID tag **12**A, **12**B, or can be any subset or field of data recognizable by position, offset, delimiter or other such field identifier. The microprocessor **46** branches at step **122** based on the determination of whether the received characteristic data string corresponds, or matches, any of the stored data strings.

If the received characteristic data string corresponds to, or matches, any of the stored characteristic data strings, the reader **10** provides an indication that the RFID tag **12**A, **12**B has been read again, activating the speaker **66** to emit a single "click" and activating or "flashing" the Red RFID related LED **84** in steps **124**, **126**, respectively. The microprocessor **46** determines in step **128**, if the reader **10** is finished reading RFID tags **12**A, **12**B, as described in detail below.

If the received characteristic data string does not correspond to, or match any of the stored data strings, the microprocessor **46** updates the characteristic data string buffer **49** containing the read characteristic data strings, for example storing the newly received characteristic data string to the buffer **49** in step **130**. The reader **10** can automatically transmit the read data in step **132**, for example to the host **23** (FIG. **1**). The reader **10** also provides an indication that a new RFID tag **12**A, **12**B has been read (e.g., read for the first time since the buffer **49** was initialized), activating the speaker **66** to emit a "beep" in step **134** and activating the Green RFID related LED **76** in step **136**. Control passes to the end of the routine **100** in step **118**.

FIG. **9** is a flowchart of a method **200** of determining when a reader **10** is finished reading. The microprocessor **46** can execute this method **200** in place of each step labeled "DONE" in the various other methods, such as at step **128** of FIG. **8** (discussed above), or in the other Figures (discussed below). As set out in the Figures, the method **200**, starting at step **202**, acts as a function or subroutine, returning a Boolean value (e.g., TRUE/FALSE, YES/NO, or DONE/NOT DONE conditions). While the method **200** could be implemented as an integral part of the other methods discussed herein, it is set out separately for ease of discussion.

At step **240**, the microprocessor **46** determines whether the trigger **20** has been released. A trigger release indicates

that the user is finished reading. If the trigger **20** has been released, the microprocessor **46** sets the Boolean value to "DONE" at step **242**, and passes control to an end of the routine **200** at step **218**, returning the appropriate Boolean value. For example, when returning to the method **100** (FIG. **8**), the condition "DONE" can cause the reader **10** to stop interrogating RFID tags **12**A, **12**B.

If the trigger **20** has not been released, the microprocessor **46** in step **244** determines whether a timeout condition has been exceeded. For example, the reader **10** can assume that all RFID tags **12**A, **12**B have been read if a new (e.g., not previously read) tag is not found after some length of time or some number of consecutive repeatedly read RFID tags **12**A, **12**B. While the length of time or number of repeated reads can be preset, the length or number of repeats can also be determined during the reading, for example as a function of RFID tag density (e.g. number of RFID tags per unit time). The microprocessor **46** can rely on an internal clock or a separate clock circuit (not shown) in measuring the timeout period. Employing RFID tag density to calculate the stopping condition "on the fly" reduces the likelihood of ending a search prematurely .

If the timeout condition is exceeded, the reader **10** considers reading to be finished, sets the Boolean value to "DONE" at step **242**, and passes control to the end of the method **200** at step **218**, producing the appropriate Boolean value for determining the next operation, such as turning the radio OFF. If the timeout condition is not exceeded, the microprocessor **46** determines whether a stop command has been received from the host **23** in step **246**. If a stop command has been received, the Boolean value is again set to "DONE" at step **242**, and control passes to the end of the method **200** at step **218**. If a stop command has not been received from the host **23**, the microprocessor **46** at step **248**, determines whether all RFID tags **12**A, **12**B have been read. If all RFID tags **12**A, **12**B have been read, the Boolean value is set to "DONE" at step **242** and control passes to the end of the method **200** at step **218**, returning the appropriate response. If all RFID tags **12**A, **12**B have not been read, the Boolean value is set to "NOT DONE" at step **250** and control passes to the end **218**, thereby returning the appropriate Boolean value.

Multi Tag Read/Write Modes

FIG. **10**, shows an additional, or alternative embodiment of operating under the present invention. Similar steps in the methods are assigned reference numerals that have the two least significant digits in common (e.g., the "Start" step is respectively numbered: **102**, **202**, **302**, . . . , **702** in FIGS. **6**–**12**, respectively).

FIG. **10** shows a method **300** of reading multiple RFID tags **12**A, **12**B (FIG. **1**) employing the reader **10** (FIGS. **1**–**3**). In a similar fashion to the method **100**, the microprocessor **46** starts executing the method **300** at step **302**, initializing the reader **10** at step **304**, turning ON the radio **44** in step **306**, and receiving responses from the RFID tags **12**A, **12**B in step **308**. In step **320**, the microprocessor **46** compares a characteristic data string from the received data to other characteristic data strings stored in the characteristic data string buffer **49** to determine whether the reader **10** has read the particular RFID tag **12**A, **12**B before. The microprocessor **46** branches at step **322** based on the determination of whether the received characteristic data string corresponds, or matches, any of the stored data strings.

If the received characteristic data string corresponds to, or matches, any of the stored characteristic data strings, the microprocessor **46** adds the read characteristic data string to the characteristic data string buffer **49**, at step **330**. The reader **10** provides an indication that the read RFID tag **12**A, **12**B has been previously read, activating the speaker **66** to emit a single "click" and activating or "flashing" the Red RFID related LED **84** at steps **352** and **354**, respectively. In step **356**, the microprocessor **46** examines a counter ("Retry") to determine whether a maximum number of iterations has been exceeded without finding a "new" (e.g. not previously read) RFID tag **12**A, **12**B. If the number of iterations without encountering a new RFID tag **12**A, **12**B has been exceeded, control passes to an end of the method **300** at step **318**. If the number of iterations without encounter a new RFID tag **12**A, **12**B has not been exceeded, the microprocessor **46** increments the Retry counter in step **358**, and determines in step **328** whether the reader **10** is finished reading RFID tags **12**A, **12**B, as described in detail above with respect to method **200** (FIG. **9**). The microprocessor **46** returns to receiving RFID tag responses in step **308**, or passes control to the end of the method **300** at step **318** based on the Boolean value returned by the method **200** (FIG. **9**).

If the received characteristic data string does not correspond to, or match any of the stored data strings, the microprocessor **46** resets the Retry counter in step **360**, and adds the read characteristic data string to the characteristic data string buffer **49** in step **362**. The reader **10** in step **364**, automatically transmits the read data, for example to the host **23**. The reader **10** also provides an indication that a new RFID tag **12**A, **12**B has been read (e.g., read for the first time since the buffer **49** was initialized), activating the speaker **66** to emit a "beep" in step **314** and activating the Green RFID related LED **76** in step **316**. The microprocessor **46** determines in step **328** whether the reader **10** is finished reading RFID tags **12**A, **12**B, as described in detail above with respect to method **200** (FIG. **9**). The microprocessor **46** returns to receiving RFID tag responses in step **308** or passes control to the end of the method **300** in step **318** based on the condition returned by the method **200**.

Inclusive Search

The reader **10** can perform an "inclusive" search, such as finding all RFID tags **12**A, **12**B on a list of RFID tags **12**A, **12**B. FIG. **11** shows a method **400** for performing an inclusive search. The user can start the inclusive search **400** by, for example, selecting an appropriate key or icon as in step **402**. The microprocessor **46** performs an initialization at step **404**, for example loading a list of characteristic data strings for the RFID tags **12**A, **12**B to be located or identified into the characteristic data string buffer **49**. The list of characteristic data strings can, for example, be downloaded from the host **23** via interface **22**. The microprocessor **46** turns ON the radio **44** at step **406**.

In step **408**, the radio **44** interrogates the RFID tags **12**A, **12**B to receive response signals containing the respective characteristic data strings. Alternatively, the radio **44** can receive the response signals without interrogating if the RFID tags **12**A, **12**B are active and periodically transmit data without requiring initiation by an interrogation signal. In step **420**, the microprocessor **46** compares the received characteristic data string with the characteristic data strings stored in the characteristic data string buffer **49**. The microprocessor **46** branches at step **422**, based on the determination of whether the received characteristic data string corresponds, or matches, any of the stored data strings.

If the read characteristic data string corresponds to, or matches any of the stored characteristic data strings, then one of the RFID tags **12**A, **12**B has been found and the reader **10** reports such to the user and/or host **23**. The reader **10** provides the user indication by activating the speaker **66** to "beep" in step **414** and activating or "flashing" the Green

RFID related LED **76** in step **416**. If the read characteristic data string does not correspond to, or match any of the stored characteristic data strings, then one of the RFID tags **12**A, **12**B has not been found, and the reader **10** reports such to the user, and/or host **23**. The reader **10** provides the user indication by activating the speaker **66** to "click" in step **424** and activating or "flashing" the Red RFID related LED **84** in step **426**.

After providing the user indications, the microprocessor determines whether the reader is finished reading, in step **428**. If the reading is finished, the returned Boolean value (i.e., DONE) causes control to pass to an end of the inclusive search routine **400** in step **418**. If the reading is not finished, the returned Boolean value (i.e., NOT DONE) causes the radio **22** to continue receiving response signals, passing control to step **418**.

Exclusive Search

The reader **10** can perform an "exclusive" search, such as finding any RFID tags **12**A, **12**B not on a list of RFID tags **12**A, **12**B. FIG. **12** shows a method **500** for performing an exclusive search. The user can start the exclusive search **500** at step **502** by, for example, selecting an appropriate key or icon. The microprocessor **46** performs an initialization at step **504**, for example loading a list of characteristic data strings for the RFID tags **12**A, **12**B to be located. At step **506**, the microprocessor turns ON the radio **44**.

In step **508**, the radio interrogates the RFID tags **12**A, **12**B to receive response signals containing the respective characteristic data strings. Alternatively, the radio can receive the response signals without interrogating if the RFID tags **12**A, **12**B are active and periodically transmit without requiring an interrogation signal. In step **520**, the microprocessor **46** compares the received characteristic data string with the characteristic data strings stored in the characteristic data string buffer **49**. The microprocessor **46** branches at step **566**, based on the determination of whether the received characteristic data string does not correspond, or match, any of the stored data strings.

If the read characteristic data string does not correspond to, or match any of the stored characteristic data strings, then one of the RFID tags **12**A, **12**B missing from the list has been found, and the reader **10** reports such to the user and/or host **23**. The reader **10** provides the user indication by activating the speaker **66** to "beep" in step **514**, and activating or "flashing" the Green RFID related LED **76** in step **516**. If the read characteristic data string corresponds to, or matches any of the stored characteristic data strings, then one of the RFID tags **12**A, **12**B missing from the list has not been found, and the reader **10** reports such to the user, and/or host **23**. The reader **10** provides the user indication by activating the speaker **66** to "click" in step **524**, and activating or "flashing" the Red RFID related LED **84** in step **526**.

After providing the user indications, the microprocessor **46** determines whether the reader **10** is finished reading, in step **528**. If the reading is finished, the returned Boolean value (i.e., DONE) causes control to pass to an end of the exclusive search routine **500** in step **518**. If the reading is not finished, the returned Boolean value (i.e., NOT DONE) causes the radio to continue receiving response signals, passing control to step **508**.

Association of RFID Tag Data With Item Using Machine-Readable Symbol

Often a user desires to make a physical association between the data read from one of the RFID tags **12**A, **12**B and a particular object or item **14** (FIG. **1**). While the RFID tag **12**A, **12**B may be attached to, or contained with the item, it can be difficult to identify the particular RFID tag **12**A, **12**B that is being read. For example, trying to identify one or more bags in a cargo hold, or cargo container on an airliner is difficult and time consuming using only RFID tags **12**A, **12**B. Each bag would have to be isolated and the RFID tag **12**A, **12**B read to ensure that the read data came from the RFID tag **12**A, **12**B associated with the particular bag. At least one proposed solution involves placing human-perceptible indicators on each of the RFID tags, as disclosed in the commonly assigned U.S. patent application Ser. No. 09/249,359, filed Feb. 12, 1999, entitled, "METHOD AND APPARATUS FOR HUMAN-PERCEPTIBLE IDENTIFICATION OF MEMORY DEVICES, SUCH AS RFID TAGS" (Atty. Docket No. 480062.663). This solution can be relatively expensive since each RFID tag **12**A, **12**B requires its own human-perceptible indicator which complicates RFID tag manufacture.

FIG. **13** shows a method **600** of associating the read data from the RFID tag **12**A, **12**B with a particular one of the items **14**. The association method **600** assumes that an RFID tag **12**A, **12**B has already been read, a characteristic data string retrieved and stored, for example, in the characteristic data string, buffer **49**. The user can start the association method **600** in step **602**, as discussed generally above. Alternatively, the reader **10** can be configured to automatically start the association method **600** at step **602**. In step **668**, the microprocessor **46** enters the symbol reading, mode. The user activates the trigger **20** in step **670**, causing the microprocessor **46** to activate the image sensor **52** to read the machine-readable symbol **24**A, **24**B at which the reader **10** is directed. In step **672**, the image sensor **52** acquires data from the machine-readable symbol **24**A, **24**B by scanning, digitizing, or by any commonly known methods in the relevant art. As part of acquiring the data, the microprocessor **46**, or a dedicated processor (not shown), decodes the image to acquire a characteristic data stringy encoded in the machine-readable symbol **24**A, **24**B. Methods and apparatus for acquiring data from machine-readable symbols are commonly known in the art, and are specifically taught in *The Bar Code Handbook* 3rd ED., by Palmer, Roger C, Helmers Publishing, Inc. (ISBN 0-911261-09-5), and, in the interest of brevity, will not be described in further detail.

To determine whether the machine-readable symbol **24**A, **24**B that the reader **10** is pointing at is associated with the RFID tag data read by the reader **10**, the microprocessor **46** compares a characteristic data string read from the RFID tag, **12**A, **12**B with the characteristic data string read from the machine-readable symbol **24**A, **24**B, in step **620**. The user can visually associate the RFID tag **12**A, **12**B with the machine-readable symbol **24**A, **24**B since the RFID tag **12**A includes the machine-readable symbol **24**A, or the RFID tag **12**B and machine-readable symbol **24**B are carried by the same item **14**, or can be visually associated is some other manner. The user can therefore determine that the data is from a particular RFID tag **12**A, **12**B when a match is indicated by the reader **10**.

If the characteristic data string from the machine-readable symbol **24**A, **24**B corresponds to, or matches, the characteristic data string from the RFID tag **12**A,**12**B, the reader **10** provides an indication that an association exists. To provide the indication, the microprocessor **46** activates the speaker **66** to emit a single "beep" in step **614** and activates or "flashes" the Green RFID related LED **76** and the Green symbol related LED **78** in step **674**. The RFID related and the symbol related LEDs **76**, **78** are each activated, indicating that both an RFID tag **12**A, **12**B and a machine-readable symbol **24**A, **24**B have been located, providing a consistency across the user interface.

In step **676**, the microprocessor **46** can turn OFF the image sensor **52** after having found an association. In step **612**, the reader **10** can report the data, for example transmitting the RFID data to the host **23** via the communications port **38** and interface **22**. In step **676**, the reader **10** can receive a direction or command from the host **23** via the interface **22** and the communications port **38**. In step **678**, the microprocessor **46** determines whether the buffer should be modified based on the command from the host **23**. If the buffer is to be modified, the microprocessor **46** modifies the buffer at step **680**, and passes control to an end of the association method **600** in step **618**. Otherwise, the microprocessor **46** passes control directly to the end of the association method, in step **600**, without modifying the buffer.

If the characteristic data string from the machine-readable symbol **24**A, **24**B does not correspond to, or match the characteristic data string from the RFID tag **12**A,**12**B, the reader **10** provides an indication that an association does not exist. To provide the indication, the microprocessor **46** activates the speaker **66** to emit a three "Beeps" in step **682**, and activates or "flashes" the Red RFID related LED **84** and the Green symbol related LED **78** in steps **626**, **684**, respectively. The Green symbol related LED **78** is flashed to indicate that a symbol has been successfully read, while the Red RFID related **84** is flashed to indicate that the data is not associated with the machine-readable symbol **24**A, **24**B, further providing consistency across the user interface. The microprocessor **46** proceeds to the end of the method **600**, in step **618**.

Automatically Reading A Symbol Based On Proximity To RFID Tag, or Frequency of RFID Tag's Responses

FIG. **14** shows a method **700**, in which the reader **10** automatically reads the machine-readable symbol when the reader **10** is within a defined proximity of the RFID tag **12**A, and hence within the defined proximity of the machine-readable symbol **24**A. The automated symbol reading feature provides numerous benefits, for example the automated symbol reading feature can simplify operation of the reader, and/or reduce the probability of user error. The automated symbol reading feature can also reduce the amount of labor required to operate the reader **10**, and can even eliminate the need for a human operator. The method **700** of FIG. **14** can be used as part of, or with, many of the previously described methods.

The antenna **42** in the reader **10** can be directionally sensitive. The directionally sensitive antenna **42** has a directional range, in other words, the antenna is more sensitive in certain directions than other directions. As the reader **10** approaches a particular RFID tag **12**A, **12**B, that RFID tag **12**A, **12**B spends a higher percentage of time within the range of the reader **10**. In contrast, other RFID tags **12**A, **12**B are in the range a lower percentage of time. Thus, as the reader **10** comes within a predefined proximity of the RFID tag **12**A, **12**B, the number of "hits" (i.e., reading an RFID tag having a desired characteristic data string) will increase, and the number of "misses" (i.e., reading RFID tags not having the desired characteristic data string) will decrease. The user may recognize this from an increase in the number of "Beeps" and a decrease in the number of "Clicks" emitted by the reader **10**. The microprocessor **46** in the reader **10**, can keep track of the number of hits and the number of misses for some unit length of time, steps **786**, **788**, respectively. The microprocessor **46** can determine a ratio of the number of hits per unit of time and the number of misses per unit of time. Alternatively, the host **23** can process the same information.

In step **790**, the microprocessor **46** determines whether the ratio of hits to misses exceeds a symbol reading threshold. If the ratio does not exceed the symbol reading threshold, the microprocessor **46** returns to step **786** and the reader **10** continues to read the RFID tags **12**A, **12**B, continually revising and checking the ratio against the threshold.

If the ratio exceeds the symbol reading threshold, the microprocessor **46** turns the image sensor **52** ON, for example, switching from the RFID reading mode to the symbol reading mode in step **768**. The microprocessor **46** controls the image sensor **52** to image and decode the machine-readable symbol **24**A, **24**B in **772**. In step **774**, the microprocessor **46** turns the image sensor **52** OFF, thereby conserving power. In step **720**, the microprocessor **46** compares the characteristic data string from the machine-readable symbol **24**A, **24**B to the characteristic data string from the RFID tag **12**A, **12**B.

If the characteristic data string from the machine-readable symbol **24**A, **24**B corresponds to, or matches, the characteristic data string from the RFID tag **12**A, **12**B, the reader **10** provides an indication that an association exists. To provide the indication, the microprocessor **46** activates the speaker **66** to emit a single "Beep" in step **714** and activates or "flashes" the Green RFID related LED **76** and the Green symbol related LED **78** in step **774**. The RFID related and the symbol related LEDs **76**, **78** are each activated, indicating that both an RFID tag **12**A, **12**B and a machine-readable symbol **24**A, **24**B have been located, providing a consistency across the user interface.

In **712**, the reader **10** can report the data, for example automatically transmitting the RFID data to the host **23** via the communications port **38** and interface **22**. In step **776**, the reader **10** can receive a direction or command from the host **23** via the interface **22** and the communications port **38**. In step **778**, the microprocessor **46** determines whether the characteristic data string buffer **49** should be modified based on the command from the host **23**. If the buffer **49** is to be modified, the microprocessor **46** modifies the buffer at step **780**, and passes control to an end of the association method **700** at step **718**. Otherwise, the microprocessor **46** passes control directly to the end of the association method **700** at step **718** without modifying the characteristic data string buffer **49**.

If the characteristic data string from the machine-readable symbol **24**A, **24**B does not correspond to, or match the characteristic data string from the RFID tag **12**A,**12**B, the reader **10** provides an indication that the association does not exist. The microprocessor **46** activates the speaker **66** to emit three "Beeps" in step **782**, and activates or "flashes" the Green symbol related LED **78** and the Red RFID related LED **84** in steps **784** and **726**, respectively. The Green symbol related LED **78** is flashed to indicate that a symbol has been successfully read, while the Red RFID related **84** is flashed to indicate that the data is not associated with the machine-readable symbol **24**A, **24**B, further providing consistency across the user interface.

SUMMARY

The various embodiments described above can be combined to provide further embodiments. All of the above U.S. patents, patent applications and publications referred to in this specification are incorporated by reference. Aspects of the invention can be modified, if necessary, to employ systems, circuits and concepts of the various patents, applications and publications to provide yet further embodiments of the invention.

Although specific embodiments of and examples data carrier readers and reading are described herein for illustrative purposes, various equivalent modifications can be made without departing from the spirit and scope of the invention, as will be recognized by those skilled in the relevant art. The teachings provided herein of the invention can be applied to any data carrier reader, not necessarily the exemplary combination RFID tag and symbol reader generally described above.

For example, some of the structures and methods can be used with readers capable of reading only RFID tags. Some of the structures and methods can be used with readers capable of reading only machine-readable symbols. Some of the structures and methods can be suitable with readers for other data carriers, such as optical tags and touch memory devices. The methods and structures are generally applicable with other wireless memory devices, not just radio frequency, and the term RFID as used herein is meant encompass wireless memory devices operating in all ranges of the electromagnetic spectrum, not only the radio frequency portion. Similarly, the structures and methods disclosed can work with any variety of modulation techniques, including, but not limited to, amplitude modulation, frequency modulation, phase modulation and/or pulse width modulation. The structures and methods can also be applied to various machine-readable symbologies, including, but not limited to, bar codes, stacked codes, area and/or matrix codes. The image sensor **52** can be any type of image capture device, including laser scanners, one- and two-dimensional charged coupled devices, Vidicons, and the like.

These and other changes can be made to the invention in light of the above-detailed description. In general, in the following claims, the terms used should not be construed to limit the invention to the specific embodiments disclosed in the specification and the claims, but should be construed to include all apparatus and methods that operate in accordance with the claims. Accordingly, the invention is not limited by the disclosure, but instead its scope is to be determined entirely by the following claims.

We claim:

1. A method of automatically searching RFID tags, comprising:

storing a number of characteristic data strings in a buffer;

reading a respective characteristic data string from each of a number of RFID tags; and

identifying any of the RFID tags that have the respective characteristic data strings that correspond to the characteristic data strings stored in the buffer after reading the respective characteristic data strings from at least two of the number of RFID tags.

2. The method of claim **1** wherein identifying the RFID tags includes comparing at least a portion of each of the read characteristic data strings to at least one of the characteristic data strings stored in the buffer.

3. The method of claim **1**, further comprising:

producing a human-perceptible indication corresponding to the number of identified RFID tags.

4. The method of claim **1**, further comprising:

producing a human-perceptible indication for each of the identified RFID tags.

5. The method of claim **1**, further comprising:

producing a human-perceptible indication having a characteristic that varies corresponding to the number of identified RFID tags.

6. The method of claim **1**, further comprising:

producing a second human-perceptible indication each time one of the read characteristic data strings matches at least one of the characteristic data strings stored in the memory.

7. The method of claim **1**, further comprising:

producing a second human-perceptible indication if all of the characteristic data strings stored in the memory match at least a respective one the read characteristic data strings.

8. The method of claim **1**, further comprising:

relaying data from the identified RFID tags to a host computer.

9. The method of claim **1**, further comprising:

transmitting an enable signal to a first one of the RFID tags that has a respective characteristic data string that matches one of characteristic data strings stored in the memory, the enable signal comprising a command to activate a human-perceptible indicator on the first one of the RFID tags.

* * * * *

EXHIBIT D

US006812852B1

(12) **United States Patent**
Cesar

(10) **Patent No.: US 6,812,852 B1**
(45) **Date of Patent: *Nov. 2, 2004**

(54) **SYSTEM AND METHOD FOR SELECTING A SUBSET OF AUTONOMOUS AND INDEPENDENT SLAVE ENTITIES**

(75) Inventor: **Christian Lenz Cesar**, Shrub Oak, NY (US)

(73) Assignee: **Intermac IP Corp.**, Everett, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/179,481**

(22) Filed: **Oct. 27, 1998**

Related U.S. Application Data

(63) Continuation of application No. 08/646,539, filed on May 8, 1996, now Pat. No. 5,828,318, and a continuation-in-part of application No. 08/694,606, filed on Aug. 9, 1996, now Pat. No. 5,942,987, which is a continuation-in-part of application No. 08/303,965, filed on Sep. 9, 1994, now Pat. No. 5,673,037.

(51) **Int. Cl.**[7] **G08C 19/00**; G05B 23/02; G06F 13/00

(52) **U.S. Cl.** **340/825.69**; 340/3.51; 340/3.58; 710/110; 710/111

(58) **Field of Search** 340/825.69, 3.51, 340/3.52; 710/110, 111

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,970,824	A	7/1976	Walton et al.
4,636,950	A	1/1987	Caswell et al.
4,656,463	A	4/1987	Anders et al.
4,673,932	A	6/1987	Ekehian et al.
4,691,202	A	9/1987	Denne et al.
5,008,661	A	4/1991	Raj
5,124,699	A	6/1992	Tervoert et al.
5,151,684	A	9/1992	Johnsen

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP	0285419 A2	10/1988	
EP	0285419	10/1988	..... G07C/9/00
EP	0285419 A3	8/1989	
EP	0585132 A1	3/1994	
EP	0702323 A2	3/1996	
NL	8802718	6/1990	

OTHER PUBLICATIONS

A.R. Grasso, "Electronic Remote Identification—A Case Study": IEEE 1990, pp. 14–2.1 to 14–2.3.

Narain H. Gehani, "Broadcasting Sequential Processes (BSP)", 1984 IEEE Transactions on Software Engineering, vol. SE–10, No. 4, Jul. 1984, pp. 343–351.

TRIS (Texas Instruments Registration and Identification System), "Stationary Reading/Writing Unit Reference Manual", Manual No. RI–ACC–UM02–01, pp. 1–3.

Set Operations and the Laws of Set Theory, Enumeration In Set Theory, pp. 45–53.

Micro RFID Communications Protocol, Pre–Release Version 0.95, 1993 Micron Communications, Inc., pp. 1–71.

DuoProx Multiple Technology Proximity Card, Hughes Identification Devices, DP1330–L Series, pp. 1–2.

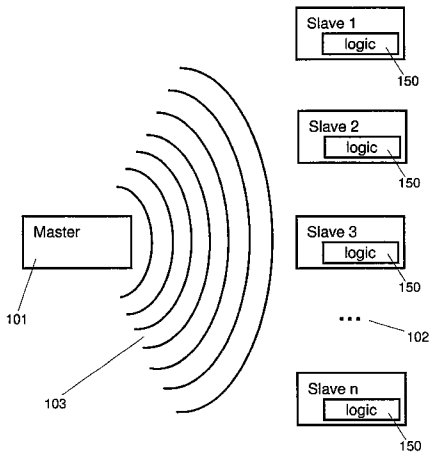
A. Nelson "Research Looks Into the Future", Think Magazine, Jul./Aug. 1994, p. 4.

Primary Examiner—Brian Zimmerman
Assistant Examiner—Clara Yang

(57) **ABSTRACT**

A master entity is capable of broadcasting commands to a plurality of three-state-selection machine slaves. Transitions from one state to another are effected on instruction from commands in a sequence of commands broadcast from the master. Slaves move to another state when they satisfy a primitive condition specified in the command. By moving slaves among the three sets, a desired subset of slaves can be isolated in one of the sets. This desired subset of slaves then can be moved to one of the states that is unaffected by commands that cause the selection of other desirable subsets of slaves.

19 Claims, 31 Drawing Sheets



U.S. PATENT DOCUMENTS

5,231,273	A		7/1993	Caswell et al.
5,245,534	A		9/1993	Waterhouse et al.
5,268,668	A		12/1993	Berube
5,294,931	A		3/1994	Meier
5,365,551	A		11/1994	Snodgrass et al.
5,374,930	A		12/1994	Schuermann
5,407,050	A		4/1995	Takemoto et al.
5,410,315	A		4/1995	Huber
5,434,572	A	*	7/1995	Smith 342/44
5,450,492	A		9/1995	Hook et al.
5,489,908	A		2/1996	Orthmann et al.
5,550,547	A		8/1996	Chan et al.
5,550,548	A		8/1996	Schuermann
5,590,339	A	*	12/1996	Chang 713/310
5,673,037	A		9/1997	Cesar et al.
5,942,987	A		8/1999	Heinrich et al.

* cited by examiner

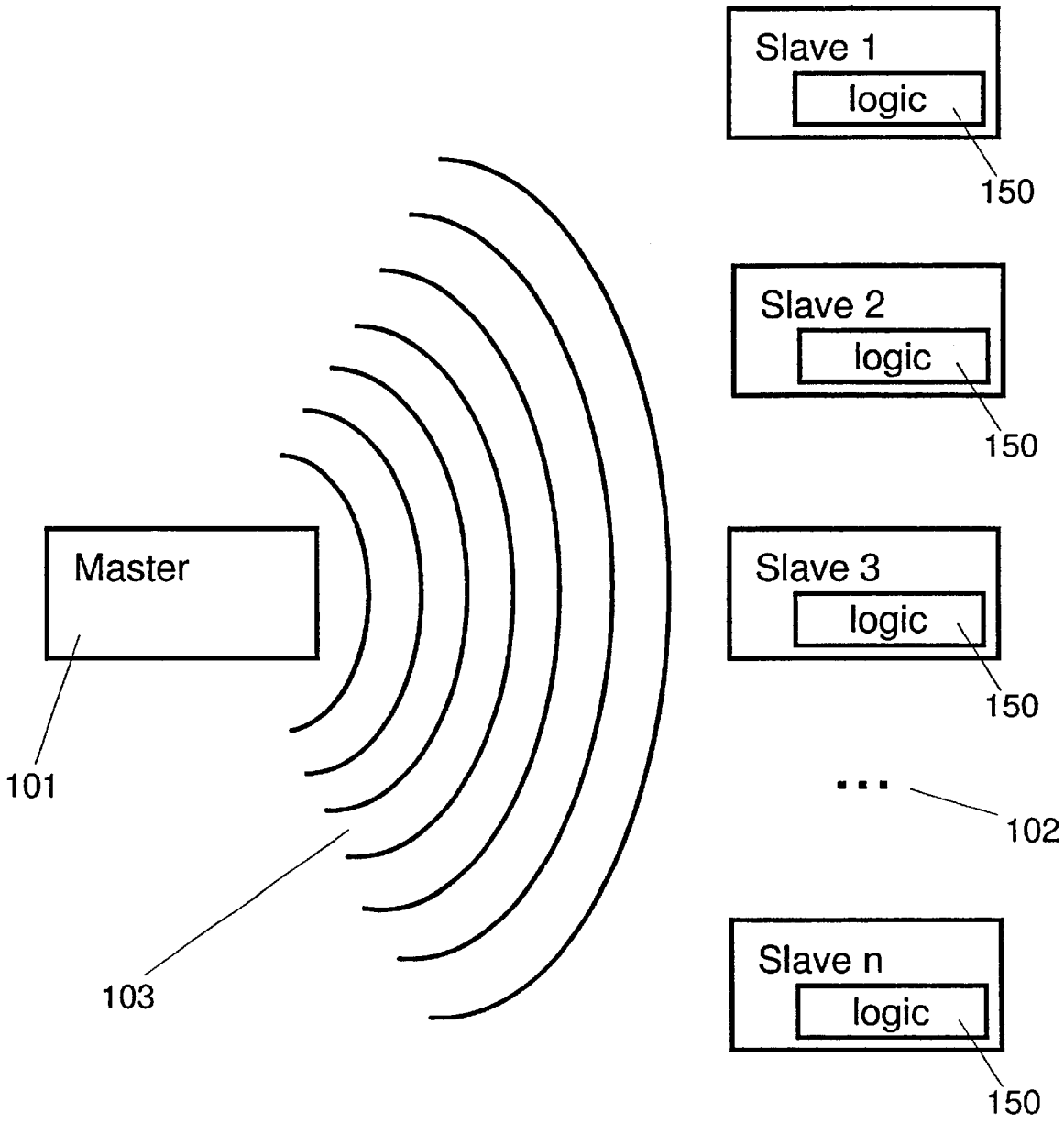

Slave 1
logic
150
Slave 2
logic
150
Master
Slave 3
logic
150
101
...
102
103
Slave n
logic
150

FIG. 1

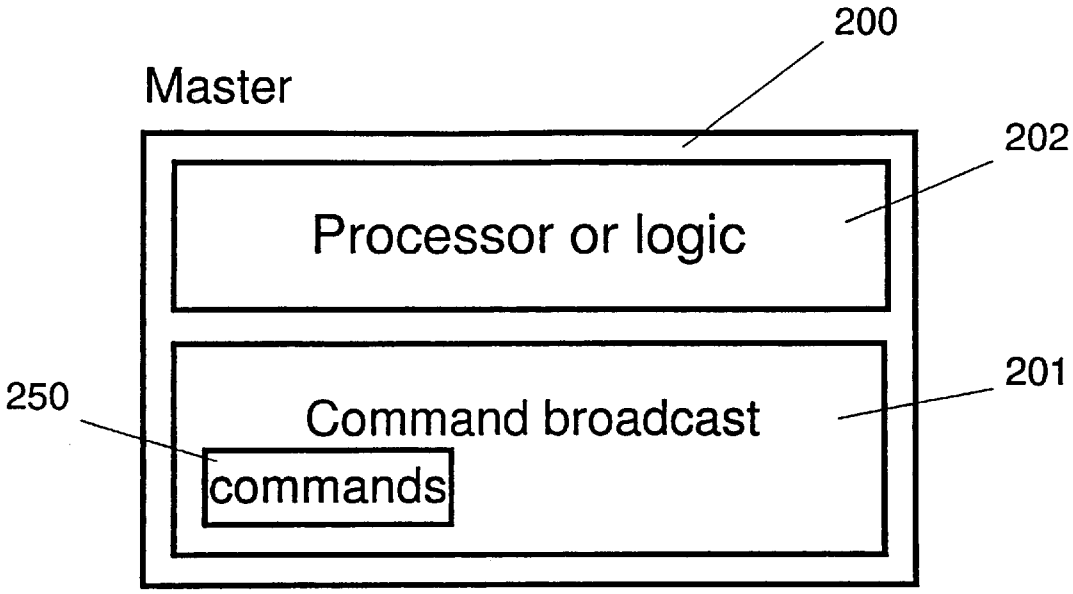

200
Master
202
Processor or logic
250
201
Command broadcast
commands

FIG. 2

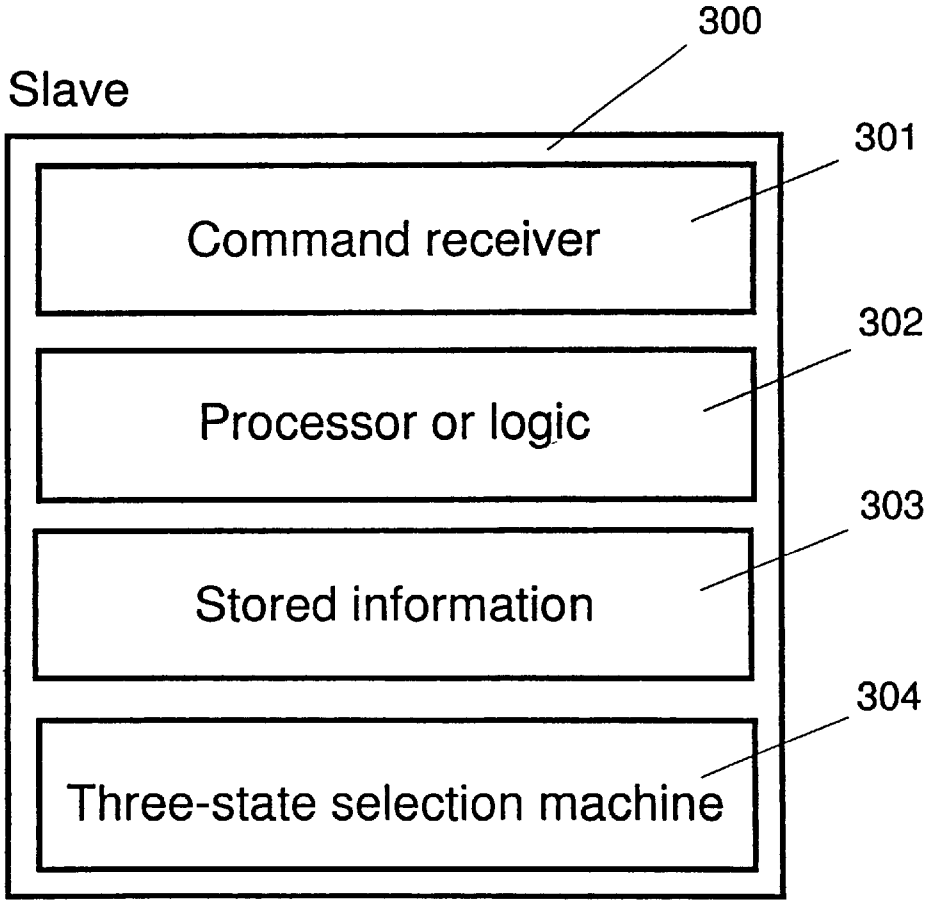

300
Slave
301
Command receiver
302
Processor or logic
303
Stored information
304
Three-state selection machine

FIG. 3

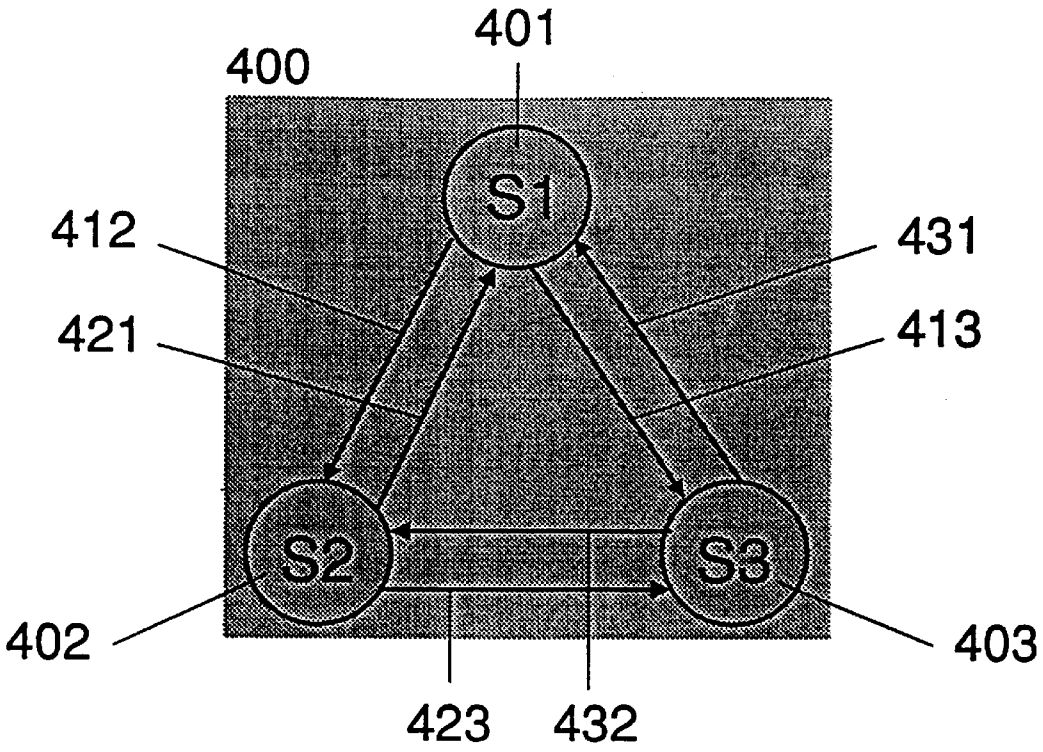

401
400
S1
412
431
421
413
S2
S3
402
403
423
432

FIG. 4

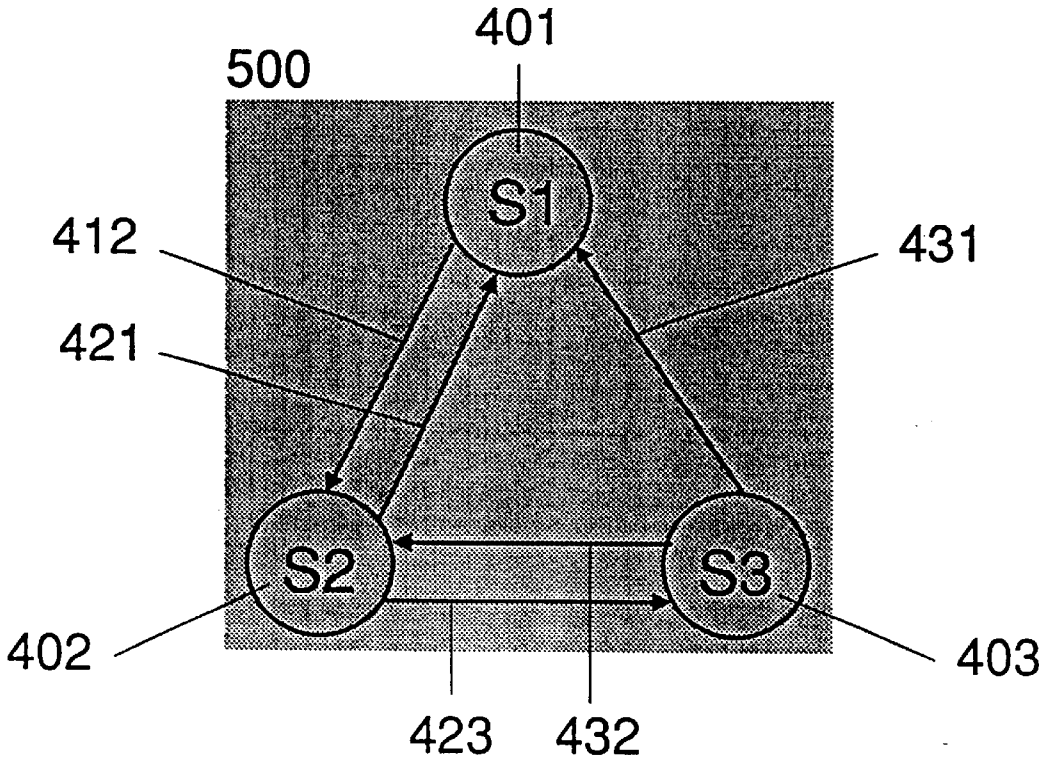

401
500
S1
412
431
421
S2
S3
402
403
423
432

FIG. 5

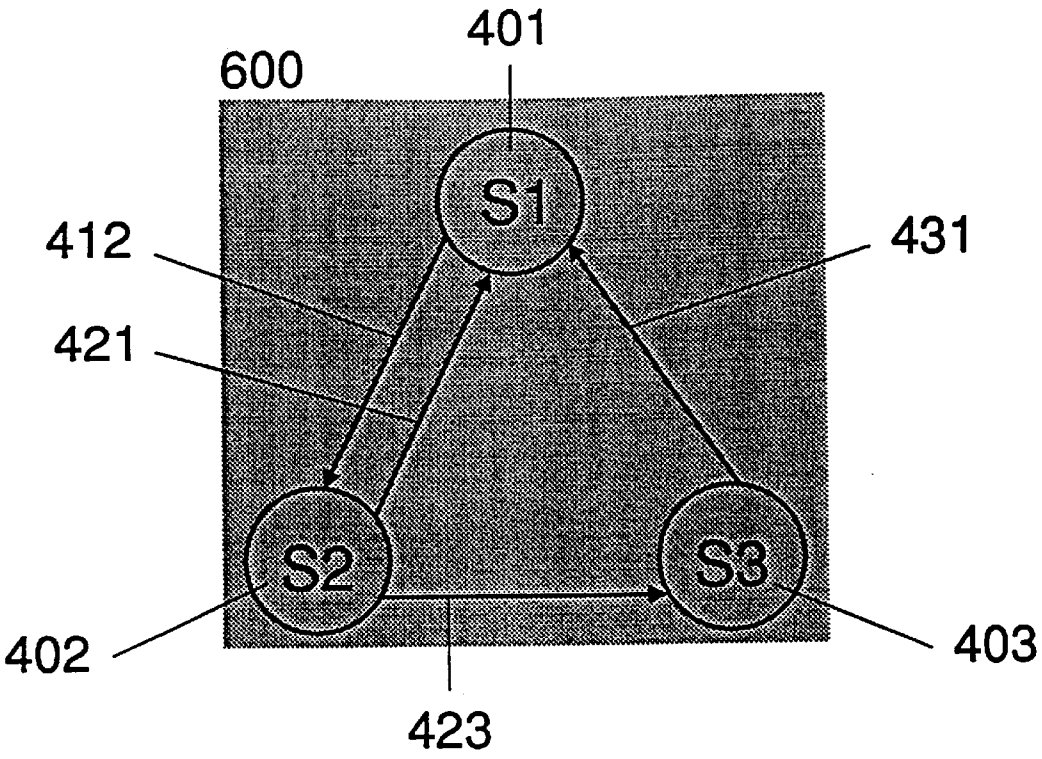

401
600
S1
412
431
421
S2
S3
402
403
423

FIG. 6

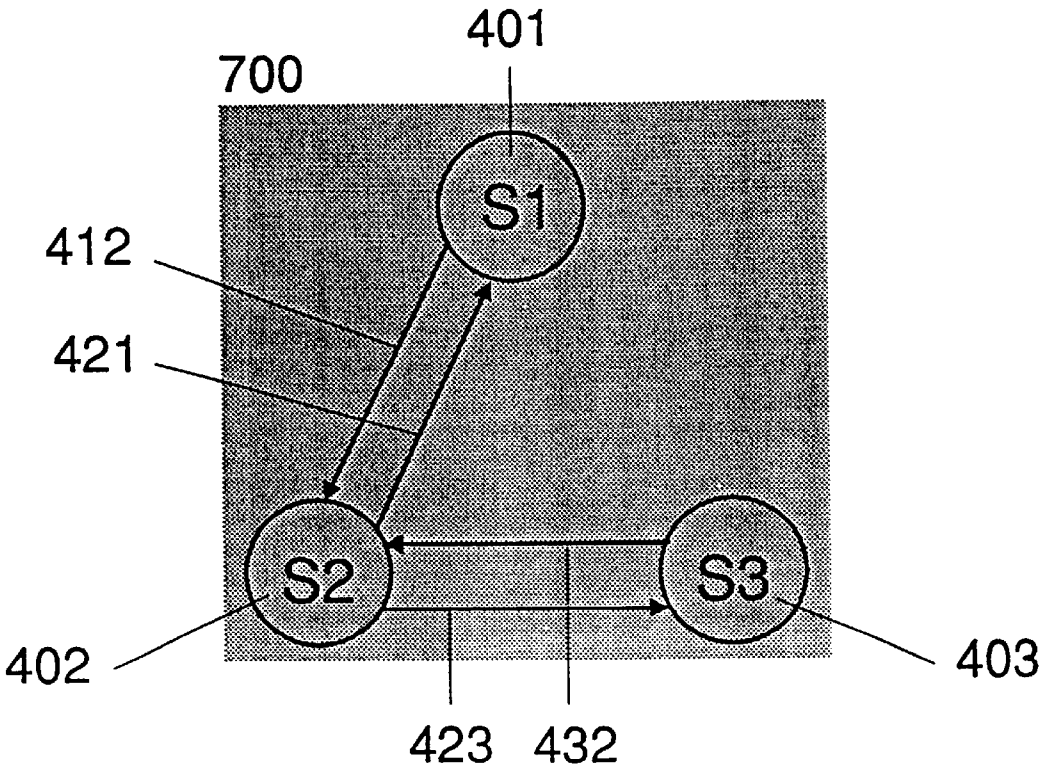

401
700
S1
412
421
S2
S3
402
403
423
432

FIG. 7

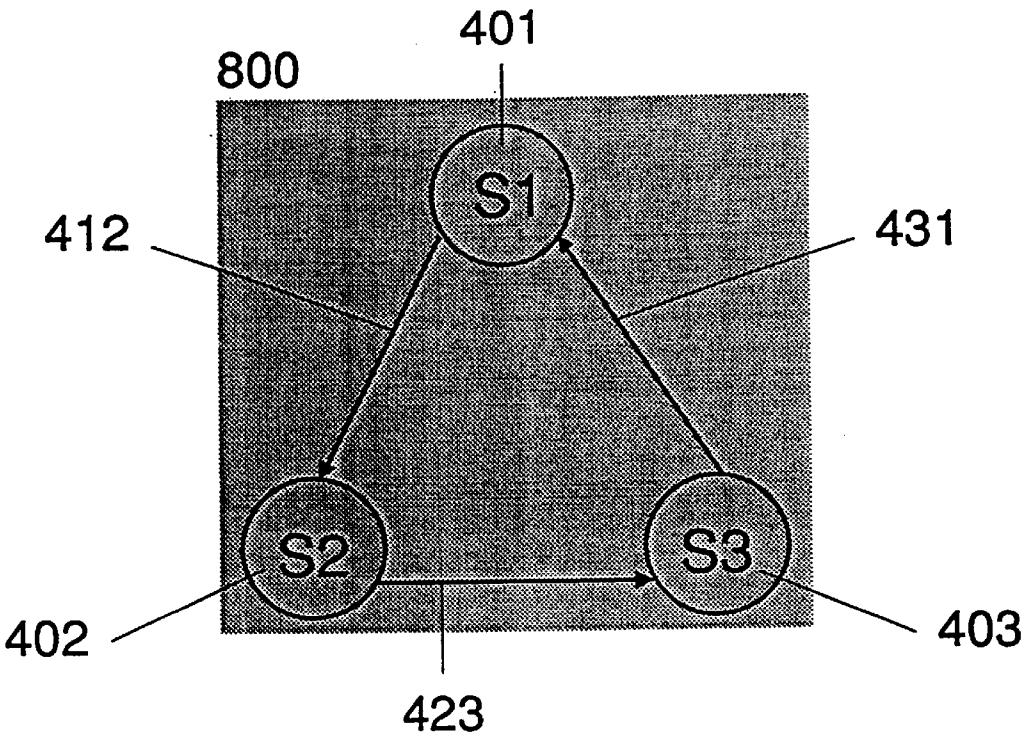


FIG. 8

	T12	T21	T23	T32	T31	T13
900	Yes	Yes	Yes	Yes	Yes	Yes
901	Yes	Yes	Yes	Yes	Yes	No
902	Yes	Yes	Yes	Yes	No	Yes
903	Yes	Yes	Yes	No	Yes	Yes
904	Yes	Yes	No	Yes	Yes	Yes
905	Yes	No	Yes	Yes	Yes	Yes
906	No	Yes	Yes	Yes	Yes	Yes
907	Yes	Yes	Yes	No	Yes	No
908	Yes	Yes	No	Yes	No	Yes
909	Yes	No	Yes	Yes	Yes	No
910	No	Yes	Yes	Yes	No	Yes
911	Yes	No	Yes	No	Yes	Yes
912	No	Yes	No	Yes	Yes	Yes
913	Yes	Yes	Yes	Yes	No	No
914	Yes	Yes	No	No	Yes	Yes
915	No	No	Yes	Yes	Yes	Yes
916	Yes	No	Yes	No	Yes	No
917	No	Yes	No	Yes	No	Yes

FIG. 9

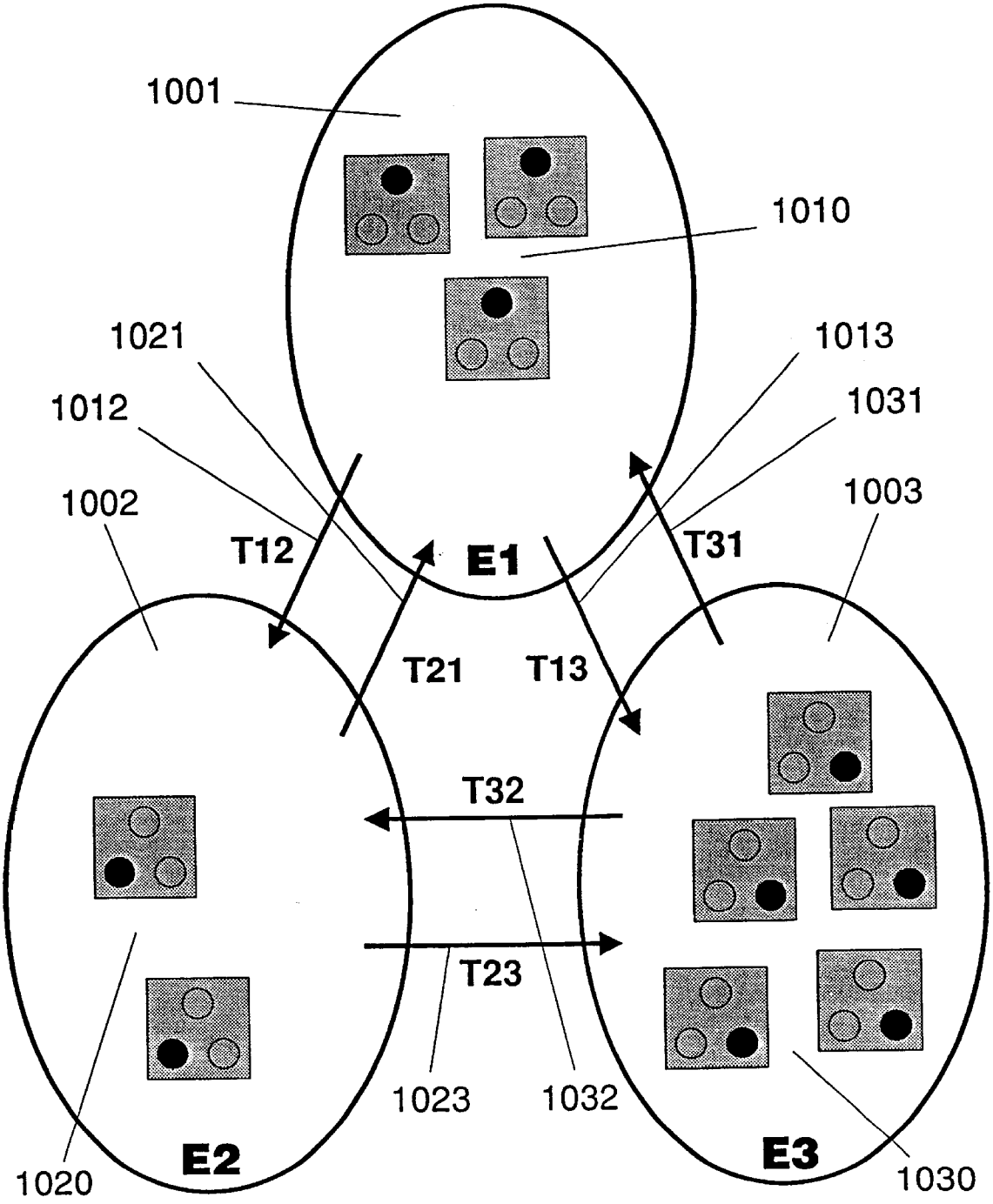

1001
1010
1021
1012
1013
1031
1002
1003
T12
E1
T31
T21
T13
T32
T23
1023
1032
E2
E3
1020
1030

FIG. 10

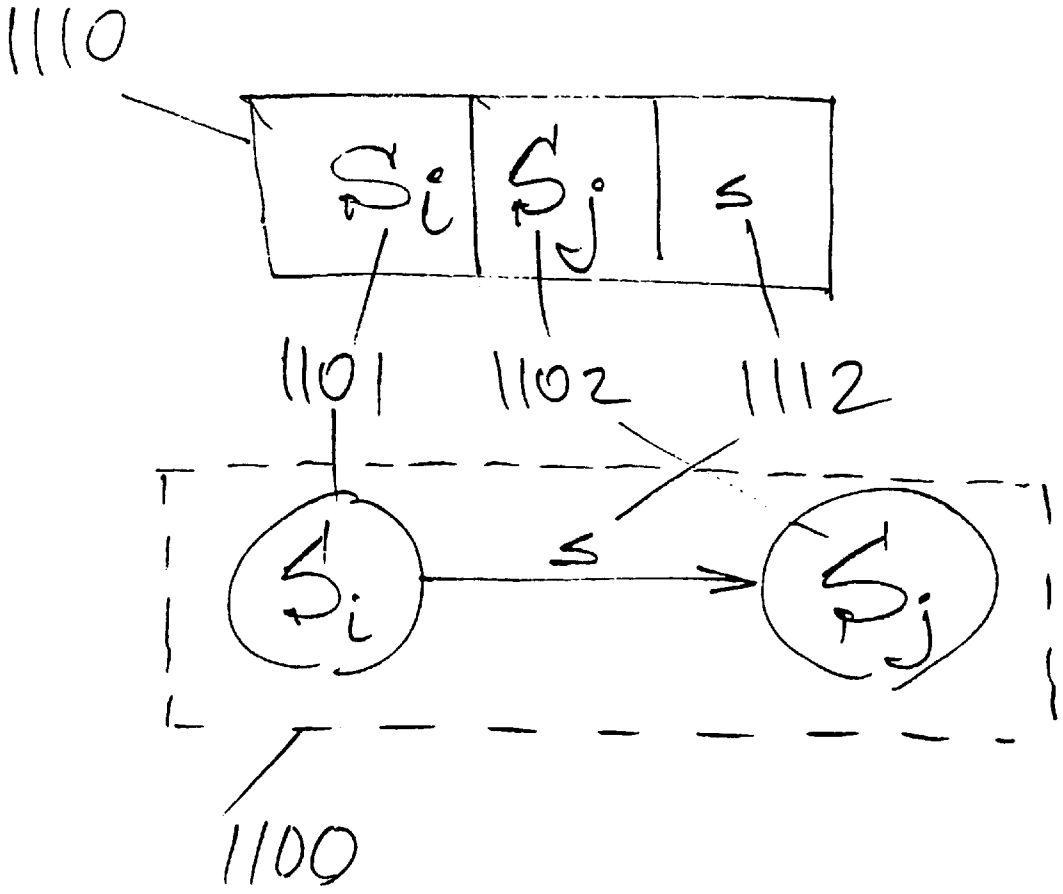


FIG. 11

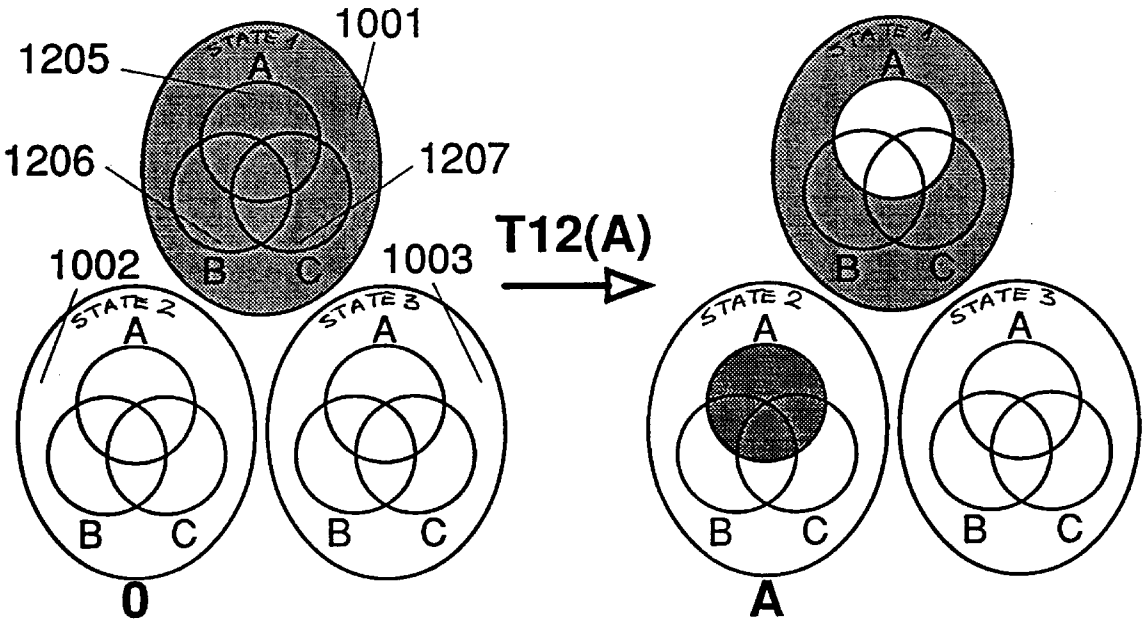

1205
1001
1206
1207
1002
1003
STATE 1
STATE 2
STATE 3
A
B
C
T12(A)
0
A

FIG. 12

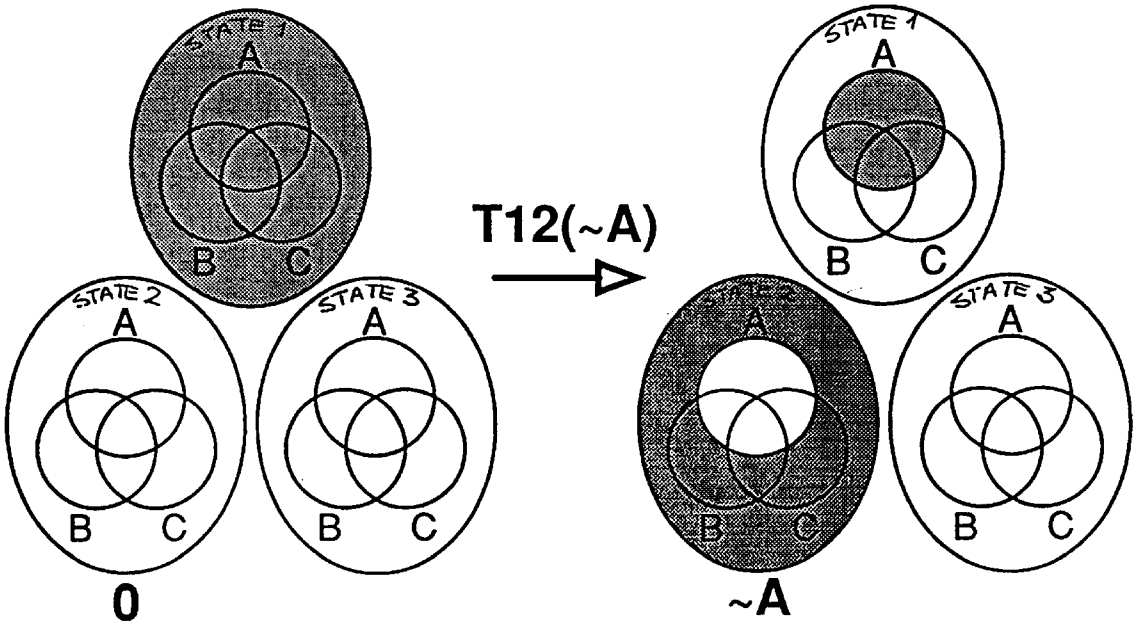

STATE 1
STATE 2
STATE 3
A
B
C
T12(~A)
0
~A

FIG. 13

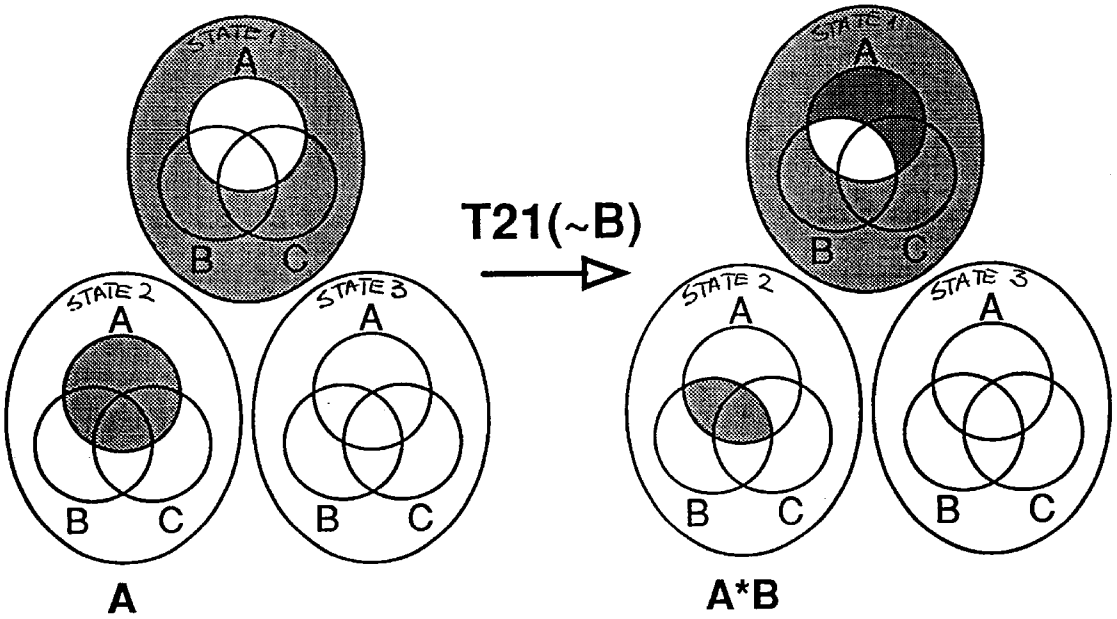

STATE 1
A
B C
STATE 2
A
B C
STATE 3
A
B C
T21(~B)
STATE 1
A
B C
STATE 2
A
B C
STATE 3
A
B C
A
A*B

FIG. 14

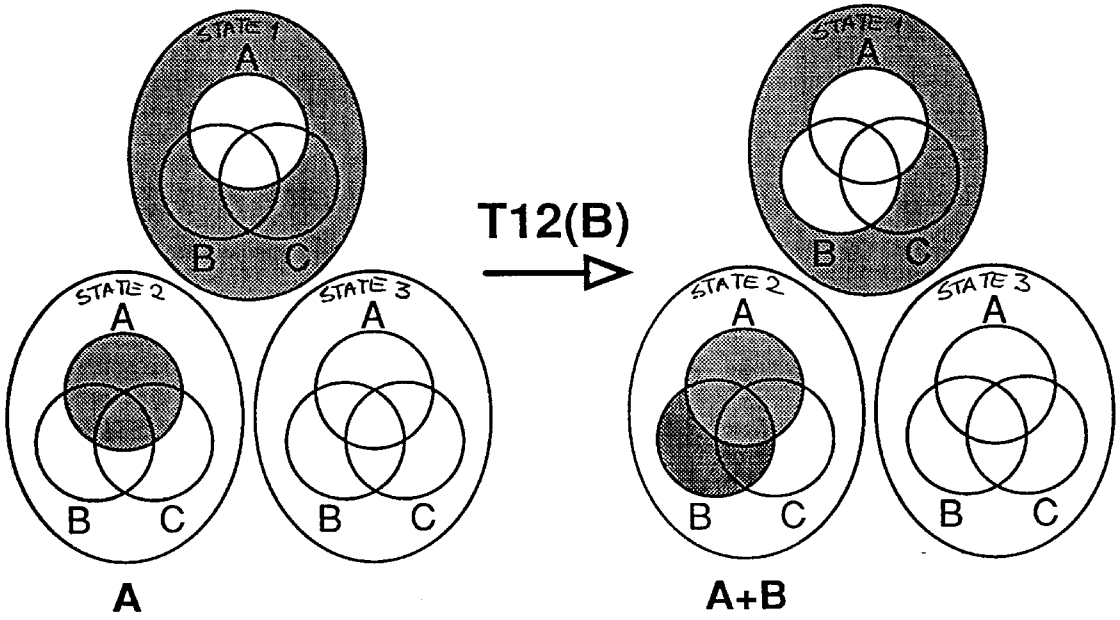

STATE 1
A
B C
STATE 2
A
B C
STATE 3
A
B C
T12(B)
STATE 1
A
B C
STATE 2
A
B C
STATE 3
A
B C
A
A+B

FIG. 15

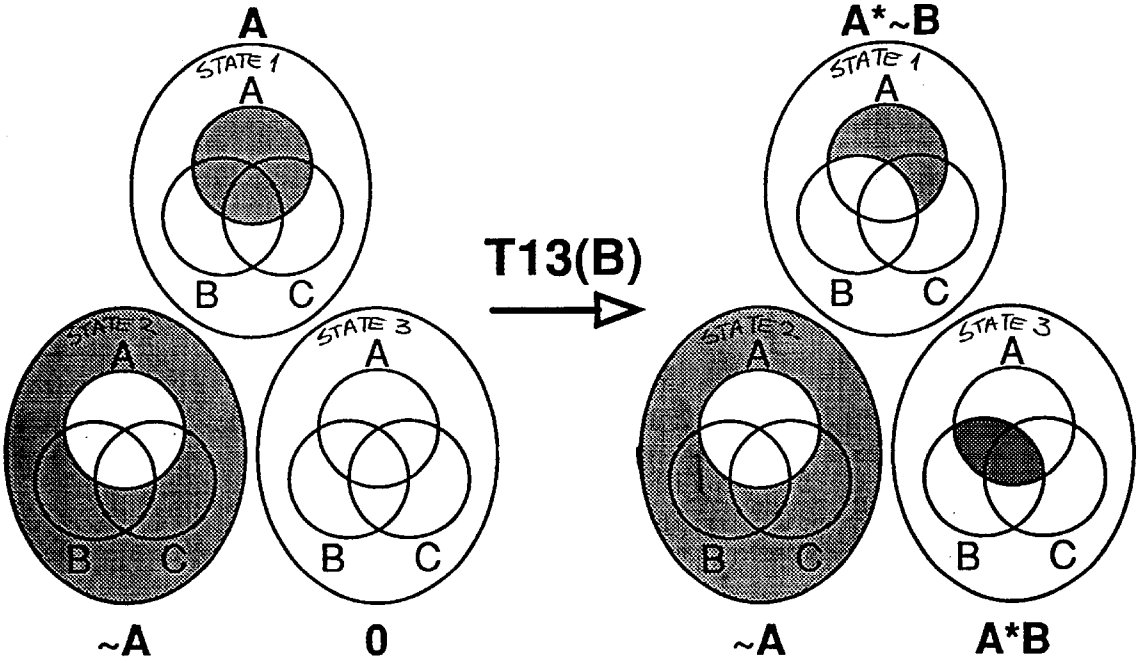

A
STATE 1
A
B C
STATE 2
A
B C
STATE 3
A
B C
~A
0
T13(B)
A*~B
STATE 1
A
B C
STATE 2
A
B C
STATE 3
A
B C
~A
A*B

FIG. 16

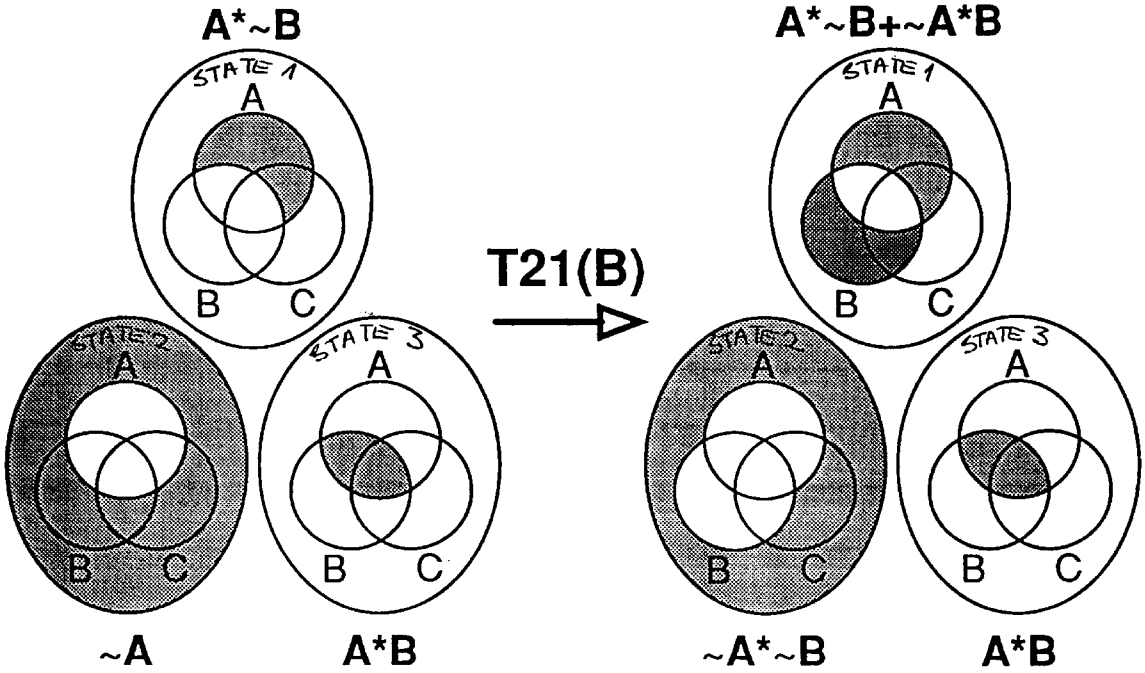
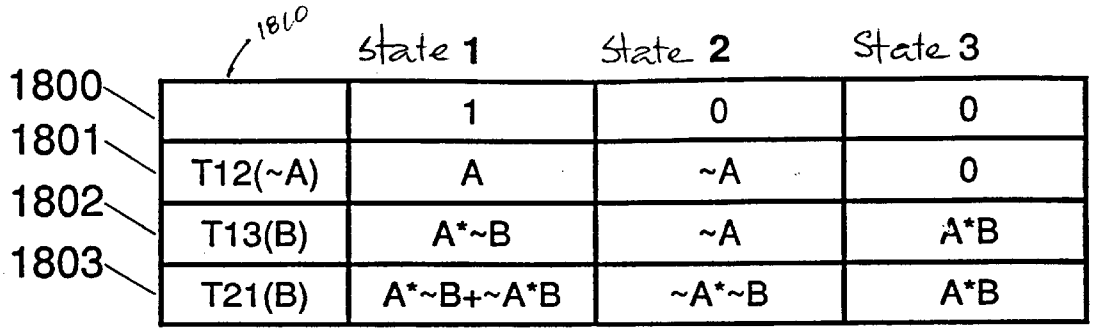
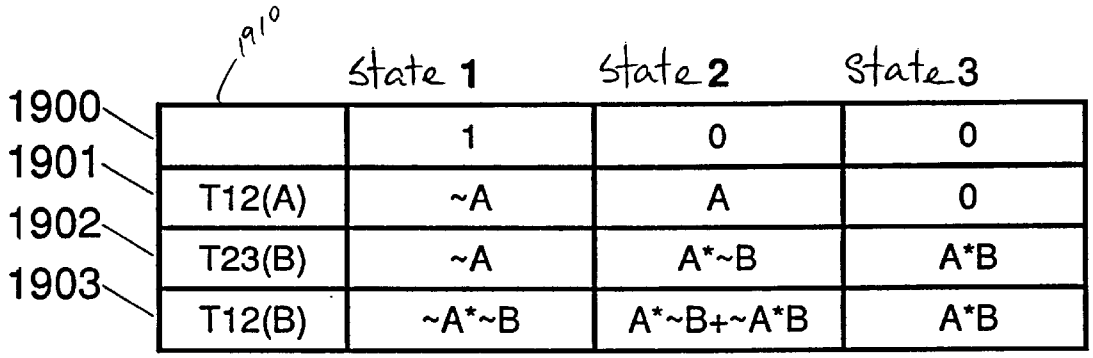

A*~B
STATE 1
A
B C
STATE 2
A
B C
STATE 3
A
B C
~A
A*B
T21(B)
A*~B+~A*B
STATE 1
A
B C
STATE 2
A
B C
STATE 3
A
B C
~A*~B
A*B

FIG. 17



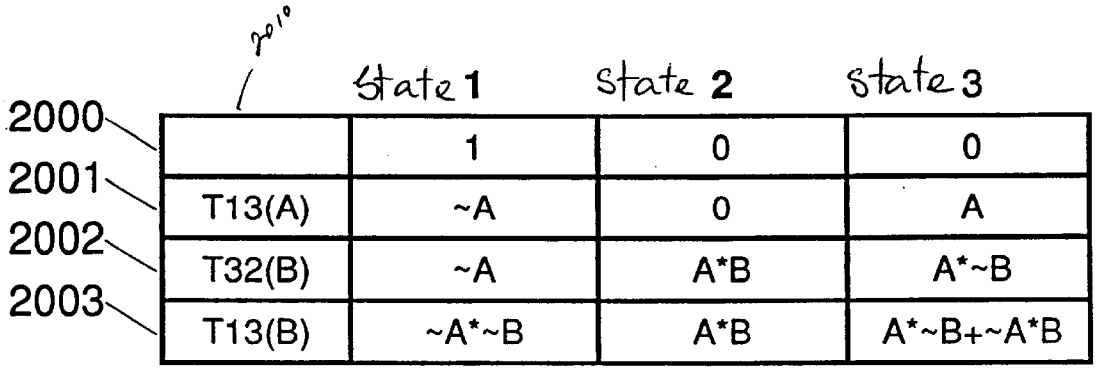
1800	State 1	State 2	State 3
	1	0	0
1801 T12(~A)	A	~A	0
1802 T13(B)	A*~B	~A	A*B
1803 T21(B)	A*~B+~A*B	~A*~B	A*B

FIG. 18



1900	State 1	State 2	State 3
	1	0	0
1901 T12(A)	~A	A	0
1902 T23(B)	~A	A*~B	A*B
1903 T12(B)	~A*~B	A*~B+~A*B	A*B

FIG. 19



2000	State 1	State 2	State 3
	1	0	0
2001 T13(A)	~A	0	A
2002 T32(B)	~A	A*B	A*~B
2003 T13(B)	~A*~B	A*B	A*~B+~A*B

FIG. 20

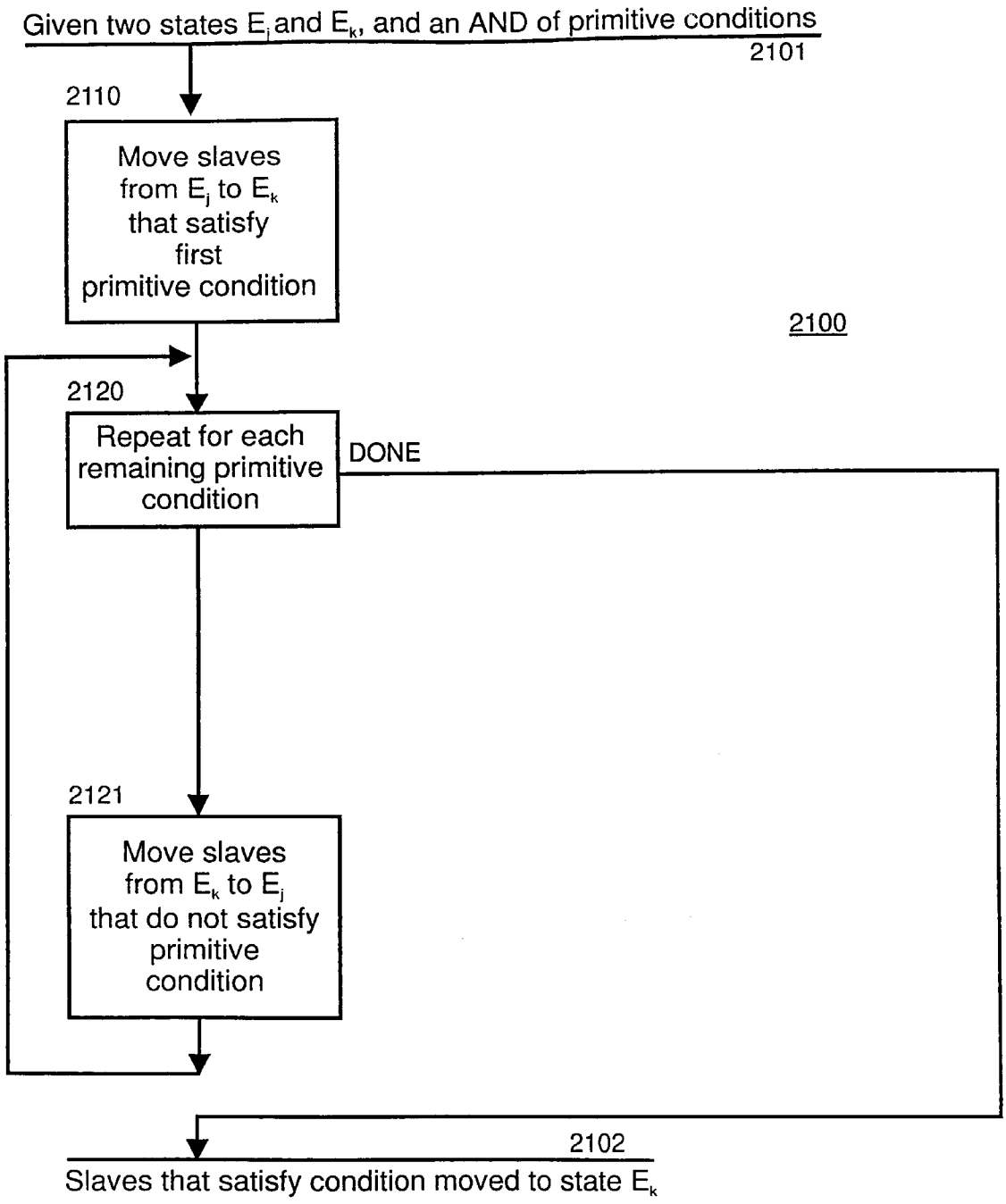

Given two states E_j and E_k, and an AND of primitive conditions
2101
2110
Move slaves from E_j to E_k that satisfy first primitive condition
2100
2120
Repeat for each remaining primitive condition
DONE
2121
Move slaves from E_k to E_j that do not satisfy primitive condition
2102
Slaves that satisfy condition moved to state E_k

FIG. 21

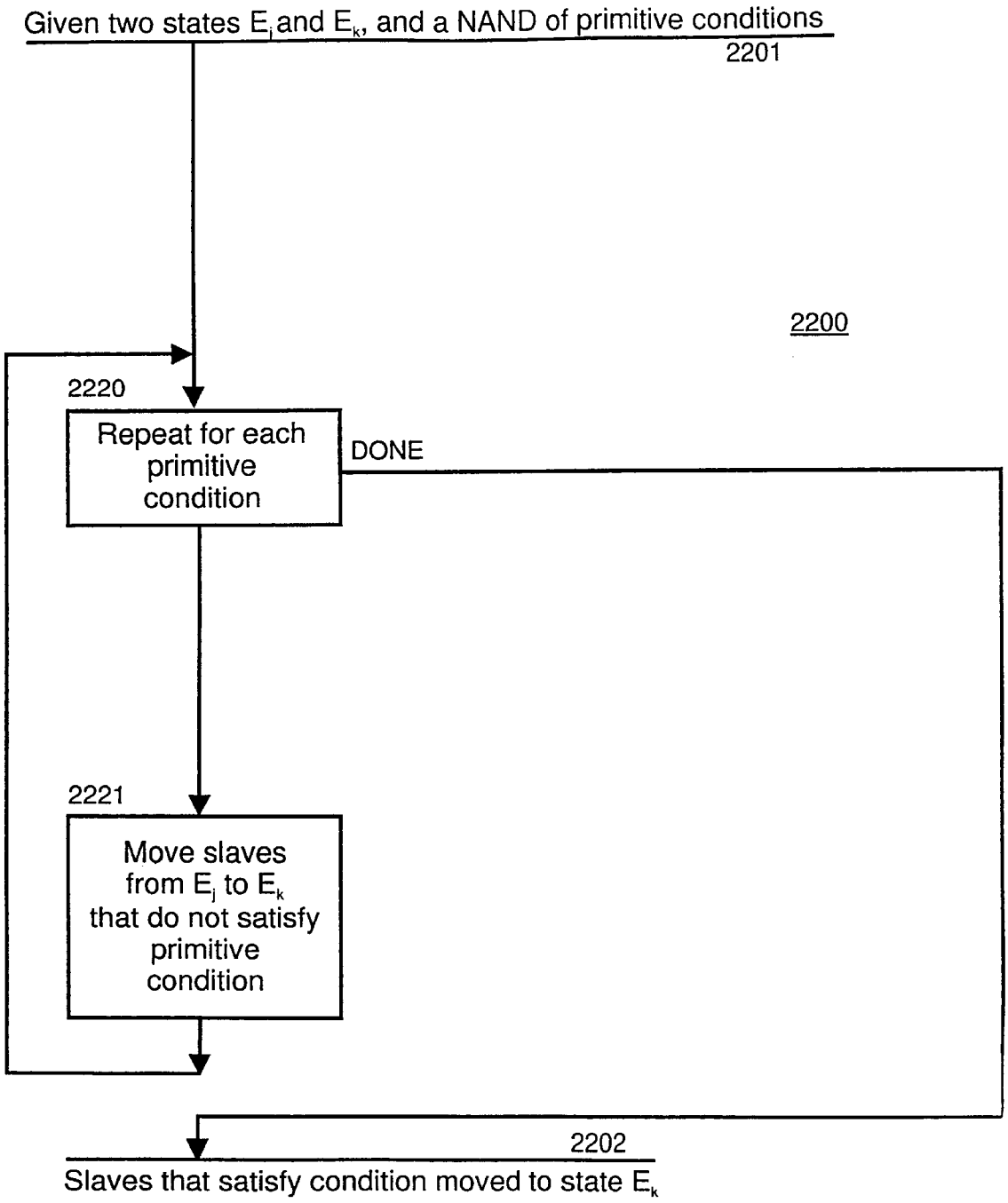

Given two states E_j and E_k, and a NAND of primitive conditions
2201
2200
2220
Repeat for each primitive condition
DONE
2221
Move slaves from E_j to E_k that do not satisfy primitive condition
2202
Slaves that satisfy condition moved to state E_k

FIG. 22

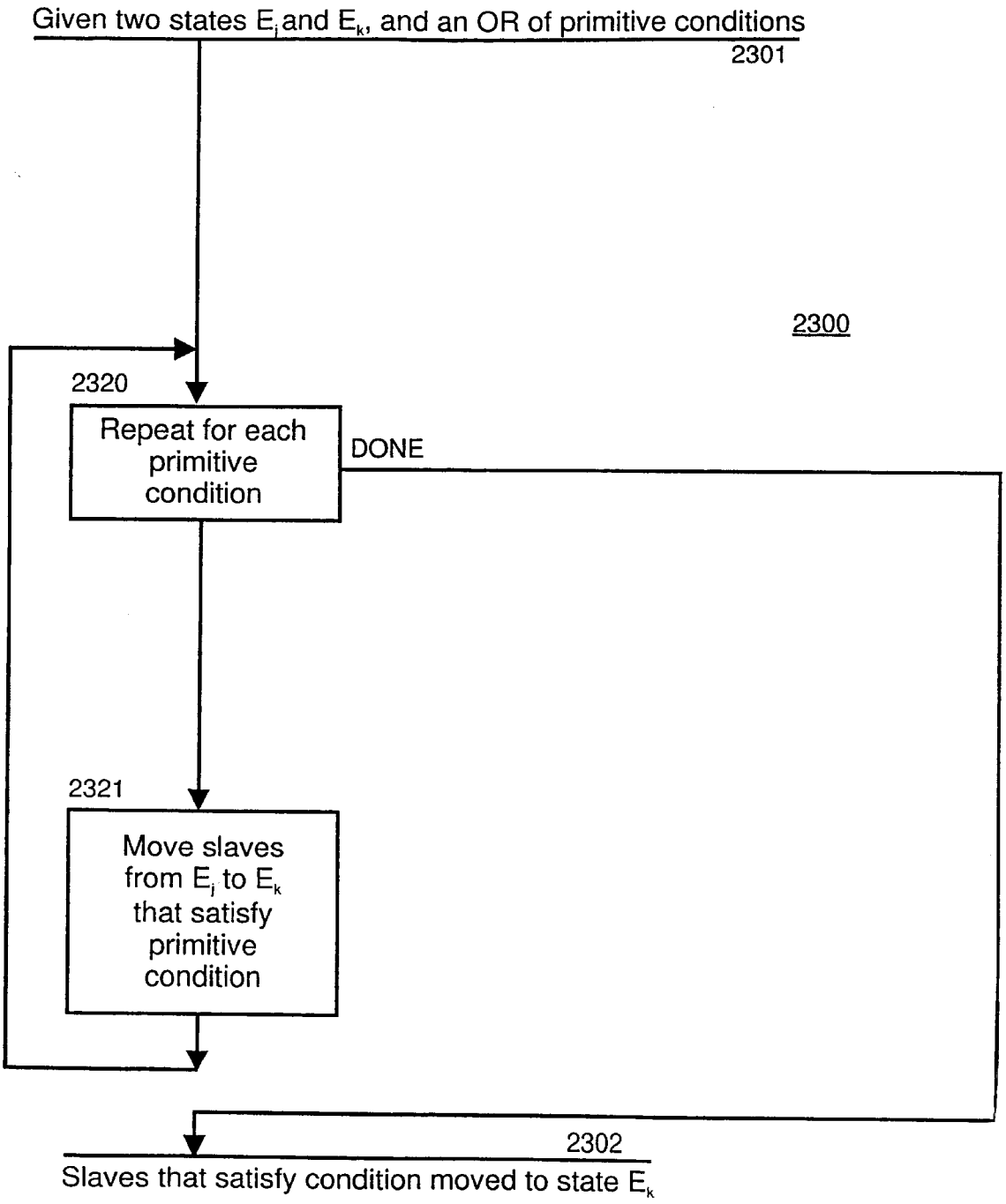

Given two states E_j and E_k, and an OR of primitive conditions
2301
2300
2320
Repeat for each primitive condition
DONE
2321
Move slaves from E_j to E_k that satisfy primitive condition
2302
Slaves that satisfy condition moved to state E_k

FIG. 23

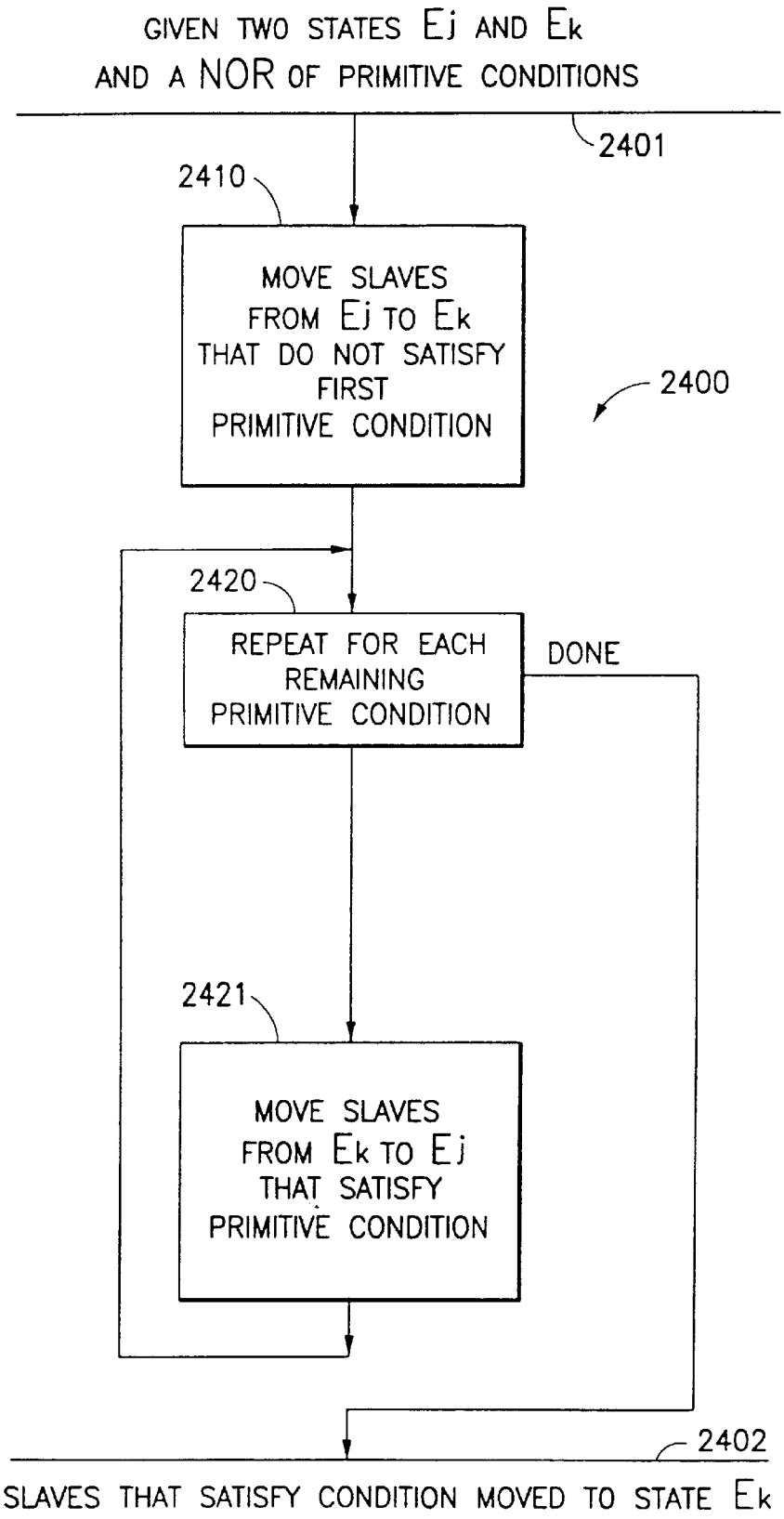

GIVEN TWO STATES Ej AND Ek
AND A NOR OF PRIMITIVE CONDITIONS
2401
2410
MOVE SLAVES FROM Ej TO Ek THAT DO NOT SATISFY FIRST PRIMITIVE CONDITION
2400
2420
REPEAT FOR EACH REMAINING PRIMITIVE CONDITION
DONE
2421
MOVE SLAVES FROM Ek TO Ej THAT SATISFY PRIMITIVE CONDITION
2402
SLAVES THAT SATISFY CONDITION MOVED TO STATE Ek

FIG. 24

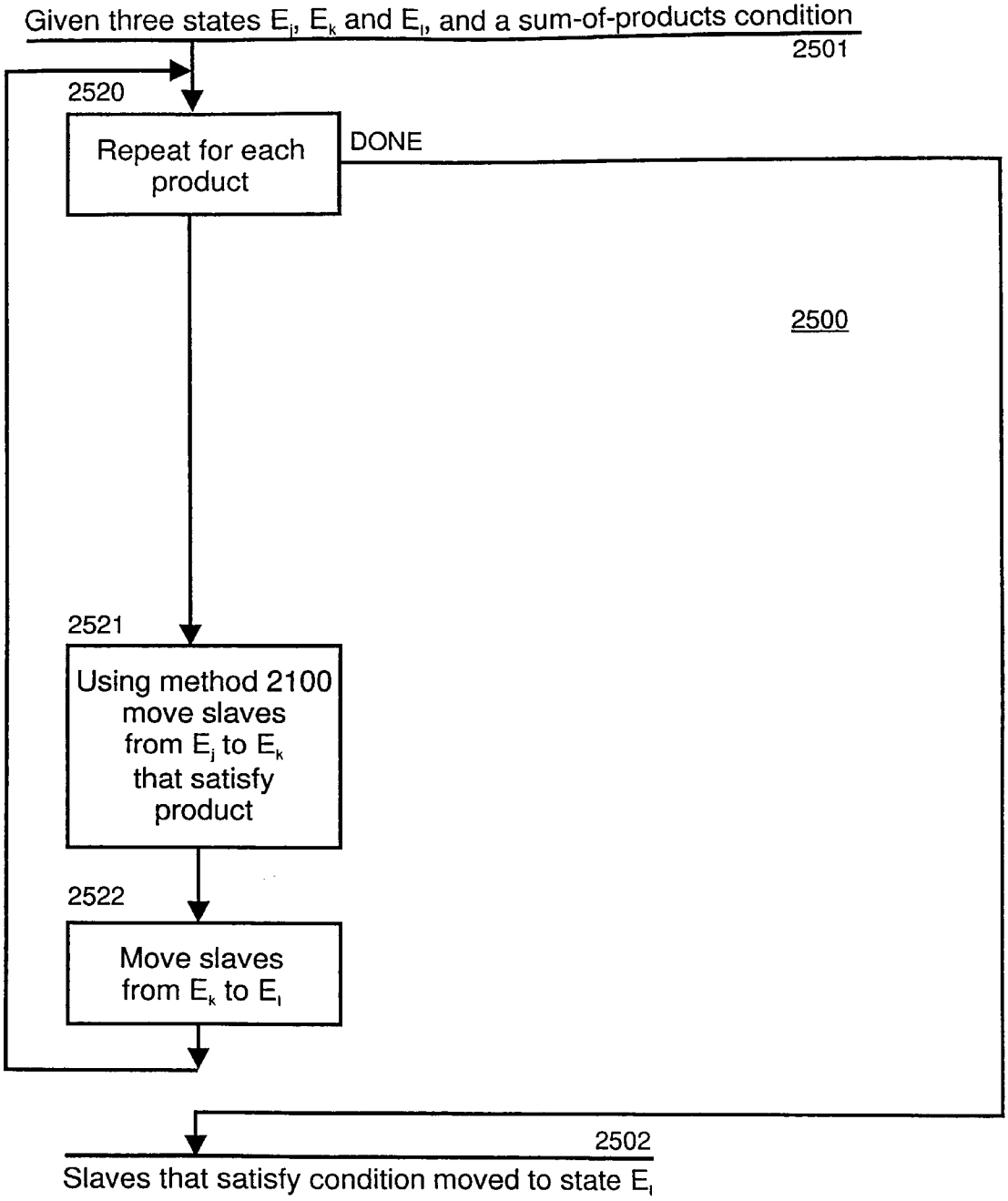

Given three states E_j, E_k and E_l, and a sum-of-products condition
2501
2520
Repeat for each product
DONE
2500
2521
Using method 2100 move slaves from E_j to E_k that satisfy product
2522
Move slaves from E_k to E_l
2502
Slaves that satisfy condition moved to state E_l

FIG. 25

	2610	STATE 1	STATE 2	STATE 3
2600		1	0	0
2601	T12(~A)	A	~A	0
2602	T21(~B)	A+~B	~A*B	0
2603	T23(1)	A+~B	0	~A*B
2604	T12(A)	~A*~B	A	~A*B
2605	T21(B)	~A*~B+A*B	A*~B	~A*B
2606	T23(1)	~A*~B+A*B	0	~A*B+A*~B

FIG. 26

	2710	STATE 1	STATE 2	STATE 3
2700		1	0	0
2701	T12(A)	~A	A	0
2702	T21(~C)	~A+~C	A*C	0
2703	T23(1)	~A+~C	0	A*C
2704	T12(B)	~A*~B+~B*~C	~A*B+B*~C	A*C
2705	T21(~C)	~A*~B+~C	~A*B*C	A*C
2706	T23(1)	~A*~B+~C	0	A*C+B*C

FIG. 27

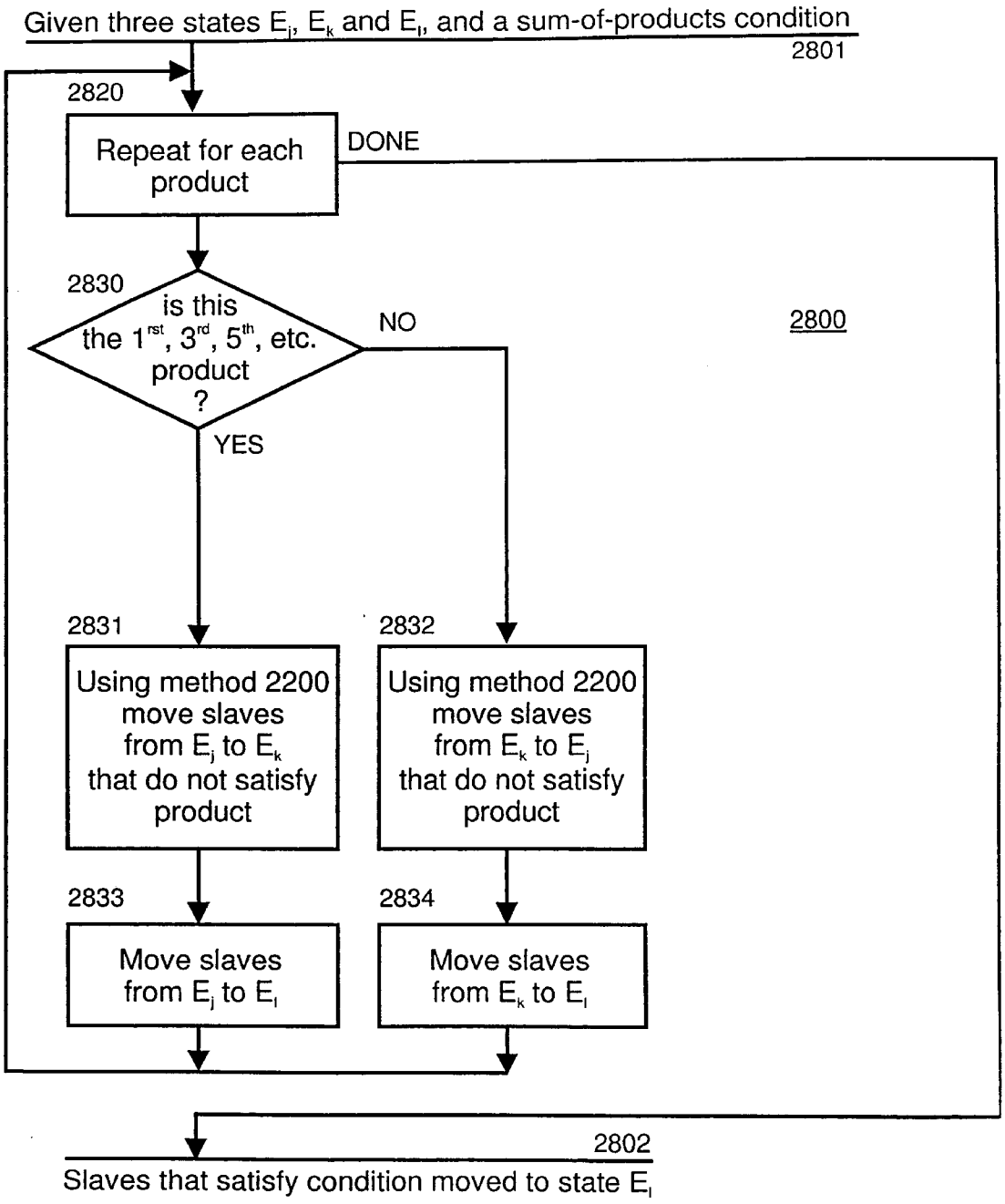
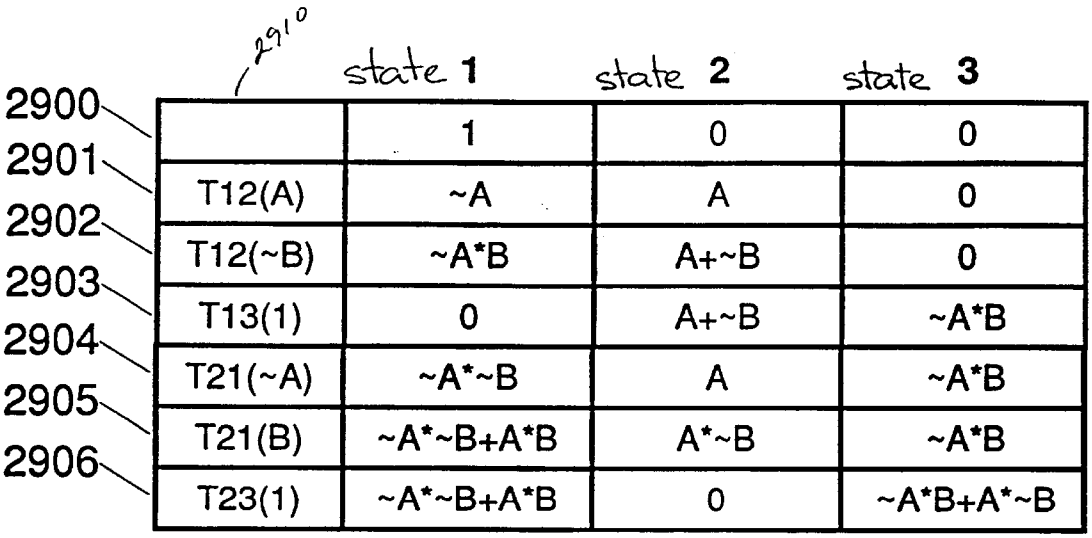
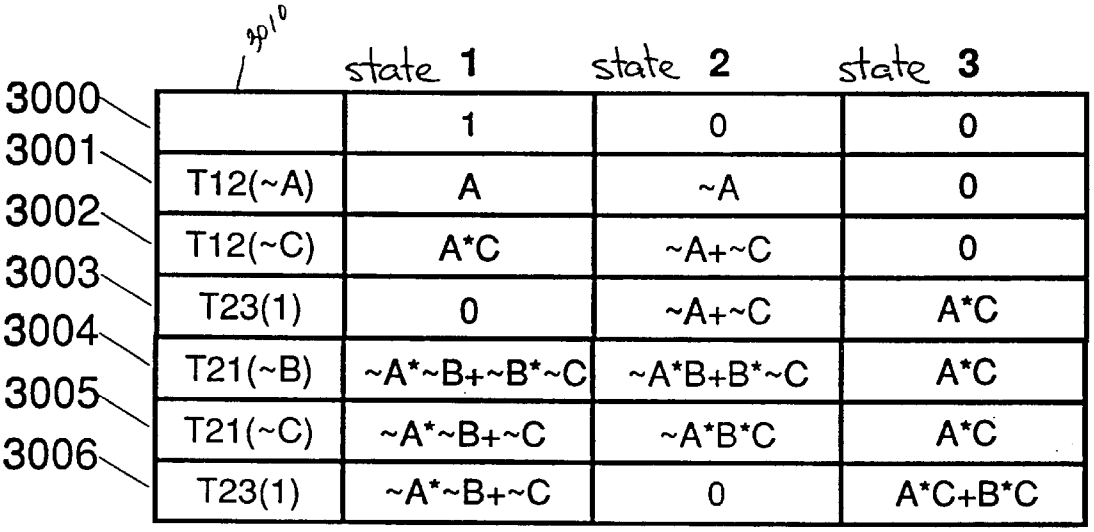

Given three states E_j, E_k and E_l, and a sum-of-products condition
2801
2820
Repeat for each product
DONE
2830
is this the 1st, 3rd, 5th, etc. product ?
NO
2800
YES
2831
Using method 2200 move slaves from E_j to E_k that do not satisfy product
2832
Using method 2200 move slaves from E_k to E_j that do not satisfy product
2833
Move slaves from E_j to E_l
2834
Move slaves from E_k to E_l
2802
Slaves that satisfy condition moved to state E_l

FIG. 28



	state 1	state 2	state 3
2900	1	0	0
2901 T12(A)	~A	A	0
2902 T12(~B)	~A*B	A+~B	0
2903 T13(1)	0	A+~B	~A*B
2904 T21(~A)	~A*~B	A	~A*B
2905 T21(B)	~A*~B+A*B	A*~B	~A*B
2906 T23(1)	~A*~B+A*B	0	~A*B+A*~B

FIG. 29



	state 1	state 2	state 3
3000	1	0	0
3001 T12(~A)	A	~A	0
3002 T12(~C)	A*C	~A+~C	0
3003 T23(1)	0	~A+~C	A*C
3004 T21(~B)	~A*~B+~B*~C	~A*B+B*~C	A*C
3005 T21(~C)	~A*~B+~C	~A*B*C	A*C
3006 T23(1)	~A*~B+~C	0	A*C+B*C

FIG. 30

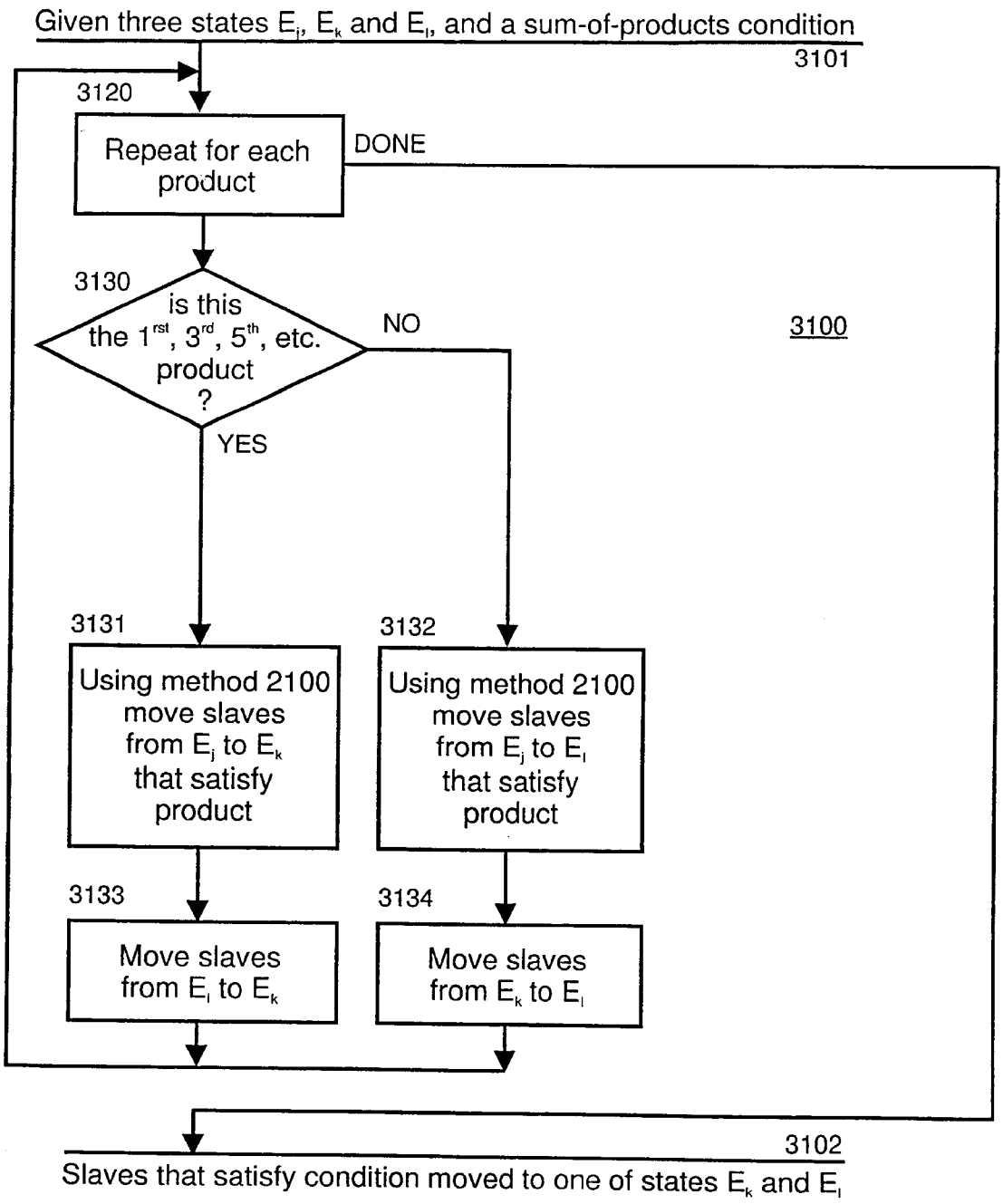
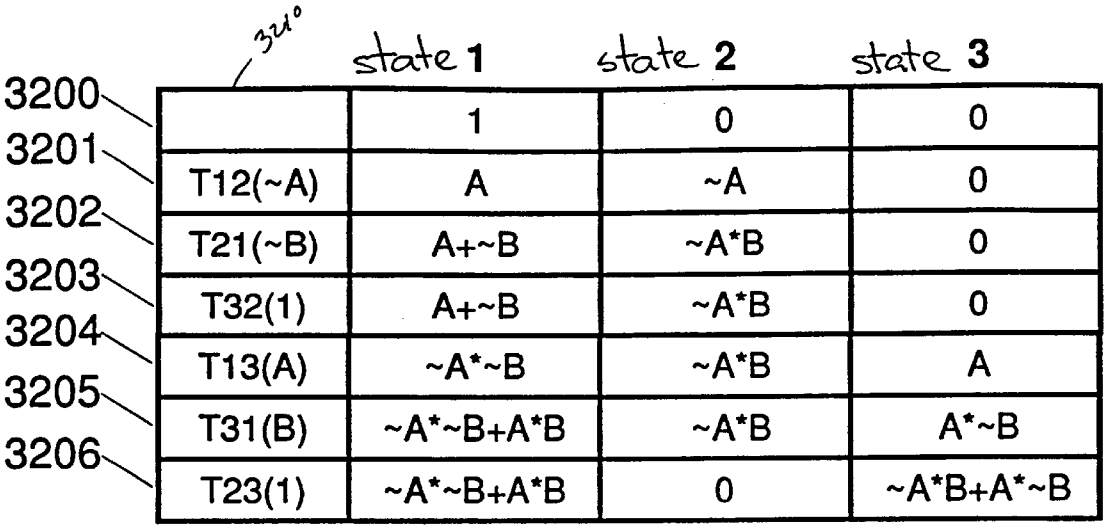
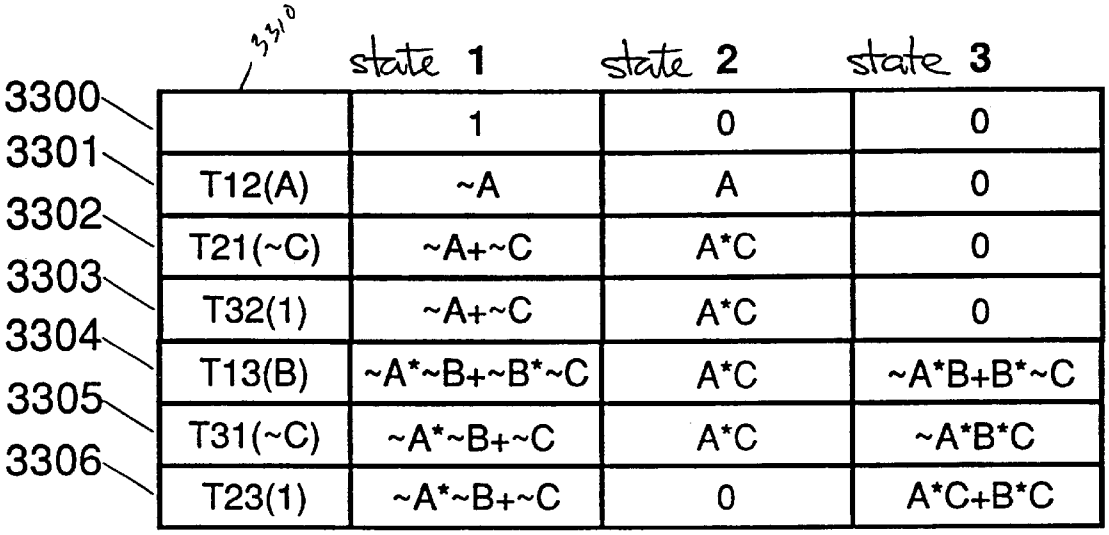

Given three states E_j, E_k and E_l, and a sum-of-products condition
3101
3120
Repeat for each product
DONE
3130
is this the 1rst, 3rd, 5th, etc. product ?
NO
3100
YES
3131
Using method 2100 move slaves from E_j to E_k that satisfy product
3132
Using method 2100 move slaves from E_j to E_l that satisfy product
3133
Move slaves from E_l to E_k
3134
Move slaves from E_k to E_l
3102
Slaves that satisfy condition moved to one of states E_k and E_l

FIG. 31



		state 1	state 2	state 3
3200		1	0	0
3201	T12(~A)	A	~A	0
3202	T21(~B)	A+~B	~A*B	0
3203	T32(1)	A+~B	~A*B	0
3204	T13(A)	~A*~B	~A*B	A
3205	T31(B)	~A*~B+A*B	~A*B	A*~B
3206	T23(1)	~A*~B+A*B	0	~A*B+A*~B

FIG. 32



		state 1	state 2	state 3
3300		1	0	0
3301	T12(A)	~A	A	0
3302	T21(~C)	~A+~C	A*C	0
3303	T32(1)	~A+~C	A*C	0
3304	T13(B)	~A*~B+~B*~C	A*C	~A*B+B*~C
3305	T31(~C)	~A*~B+~C	A*C	~A*B*C
3306	T23(1)	~A*~B+~C	0	A*C+B*C

FIG. 33

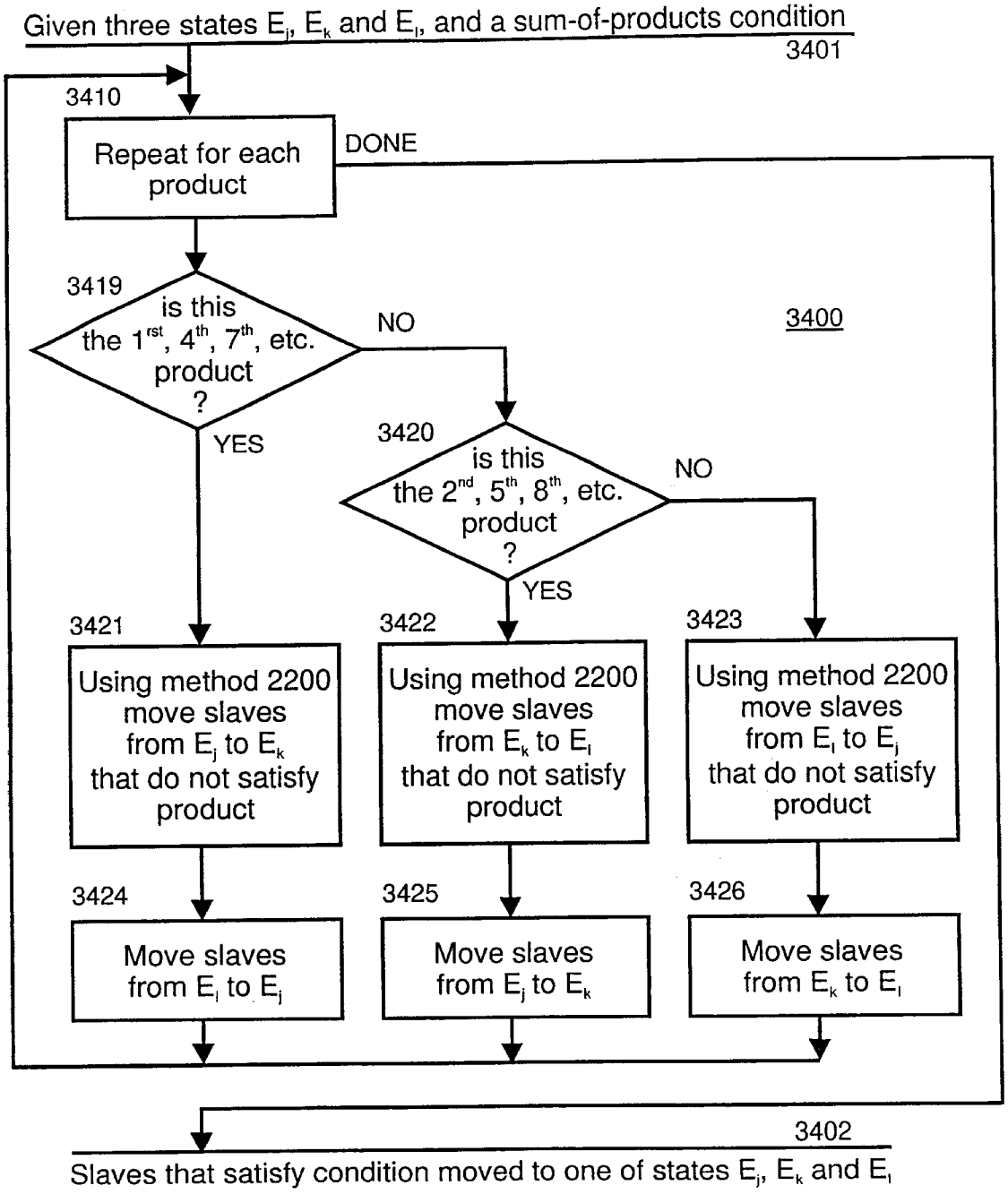
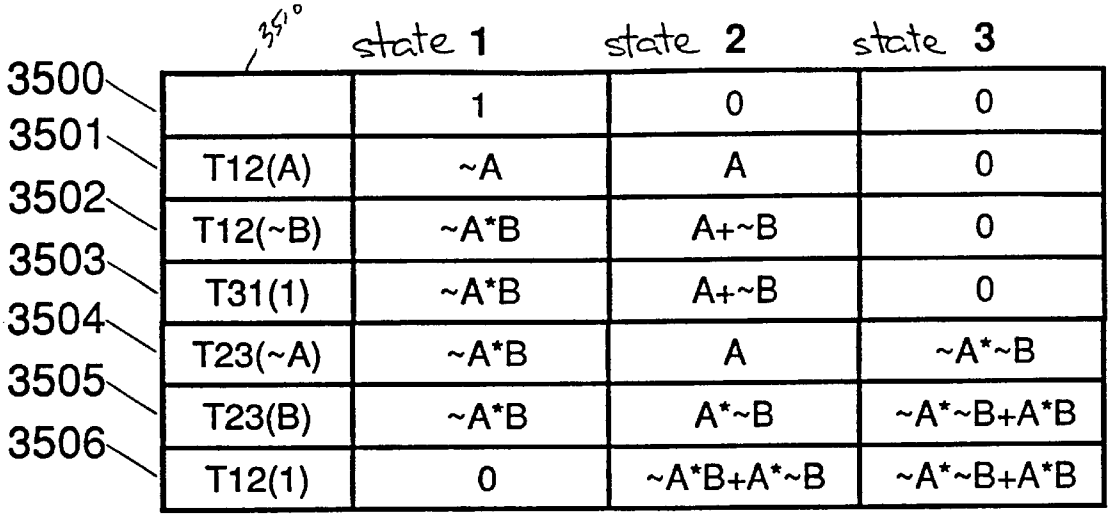
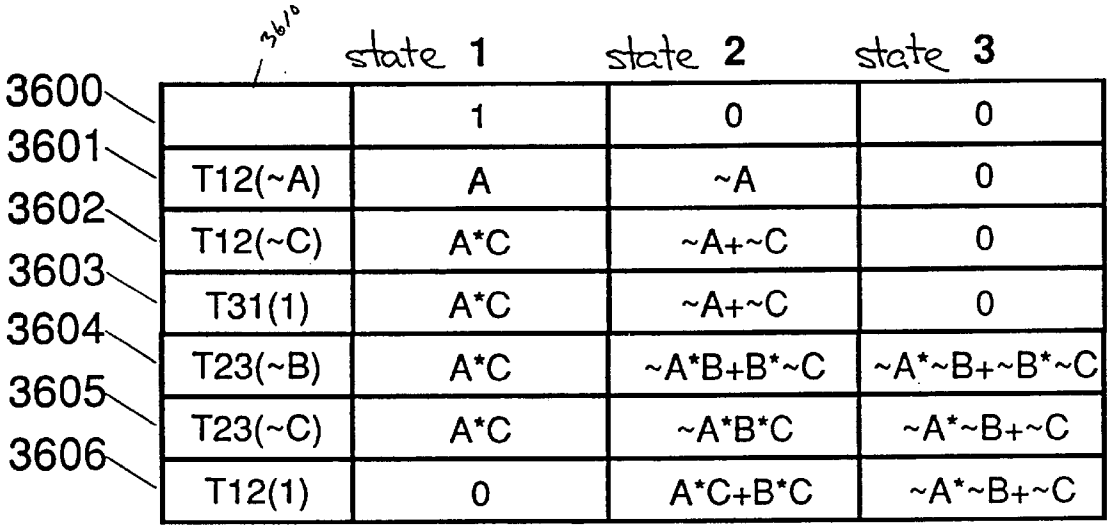

Given three states E_j, E_k and E_l, and a sum-of-products condition
3401
3410
Repeat for each product
DONE
3419
is this the 1rst, 4th, 7th, etc. product ?
NO
3400
YES
3420
is this the 2nd, 5th, 8th, etc. product ?
NO
YES
3421
Using method 2200 move slaves from E_j to E_k that do not satisfy product
3422
Using method 2200 move slaves from E_k to E_l that do not satisfy product
3423
Using method 2200 move slaves from E_l to E_j that do not satisfy product
3424
Move slaves from E_i to E_j
3425
Move slaves from E_j to E_k
3426
Move slaves from E_k to E_l
3402
Slaves that satisfy condition moved to one of states E_j, E_k and E_l

FIG. 34



	3510	state 1	state 2	state 3
3500		1	0	0
3501	T12(A)	~A	A	0
3502	T12(~B)	~A*B	A+~B	0
3503	T31(1)	~A*B	A+~B	0
3504	T23(~A)	~A*B	A	~A*~B
3505	T23(B)	~A*B	A*~B	~A*~B+A*B
3506	T12(1)	0	~A*B+A*~B	~A*~B+A*B

FIG. 35



	3610	state 1	state 2	state 3
3600		1	0	0
3601	T12(~A)	A	~A	0
3602	T12(~C)	A*C	~A+~C	0
3603	T31(1)	A*C	~A+~C	0
3604	T23(~B)	A*C	~A*B+B*~C	~A*~B+~B*~C
3605	T23(~C)	A*C	~A*B*C	~A*~B+~C
3606	T12(1)	0	A*C+B*C	~A*~B+~C

FIG. 36

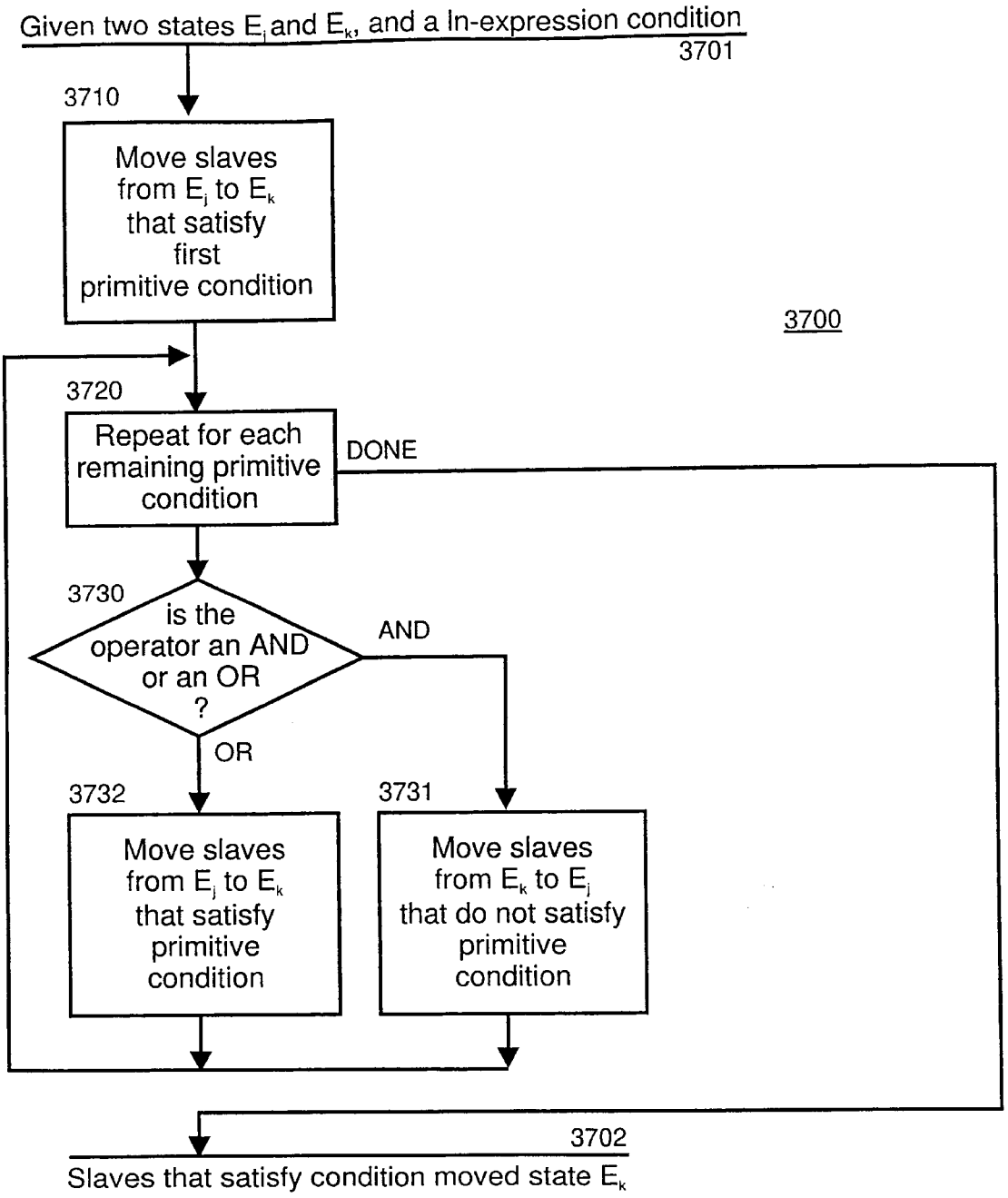

Given two states E_j and E_k, and a In-expression condition
3701
3710
Move slaves from E_j to E_k that satisfy first primitive condition
3700
3720
Repeat for each remaining primitive condition
DONE
3730
is the operator an AND or an OR ?
AND
OR
3732
Move slaves from E_j to E_k that satisfy primitive condition
3731
Move slaves from E_k to E_j that do not satisfy primitive condition
3702
Slaves that satisfy condition moved state E_k

FIG. 37

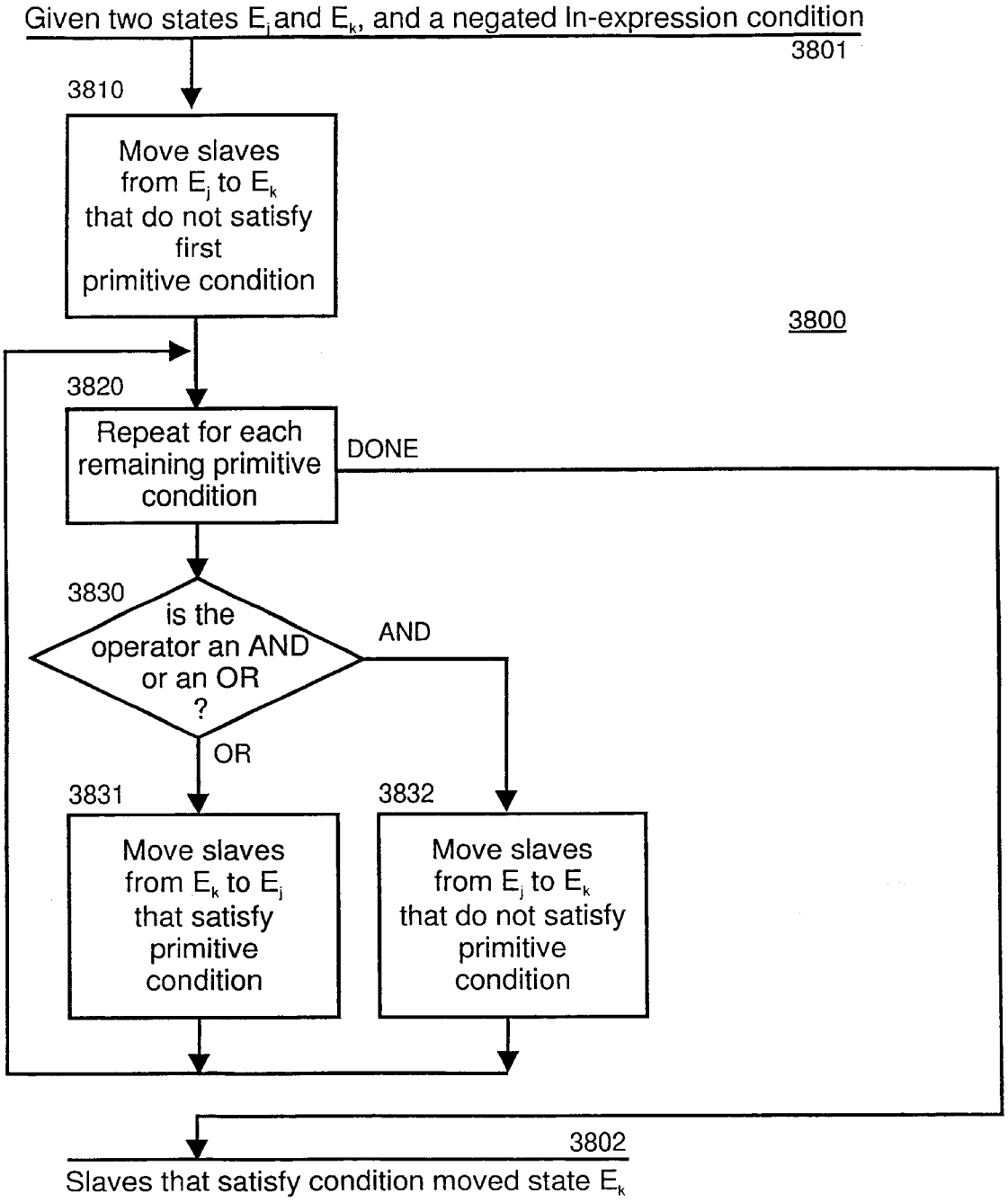
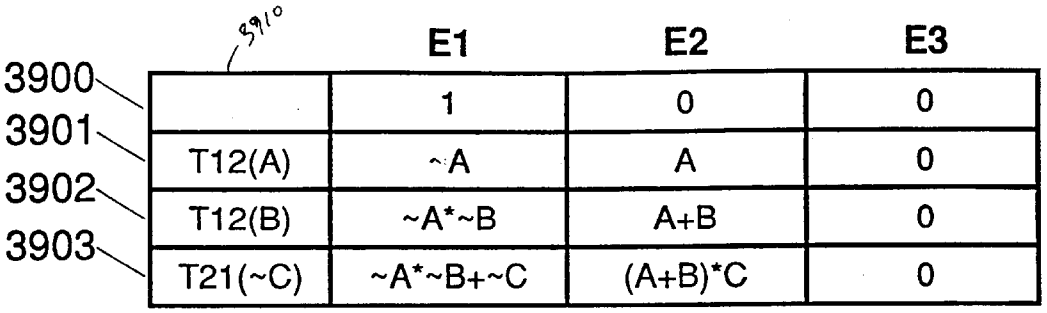
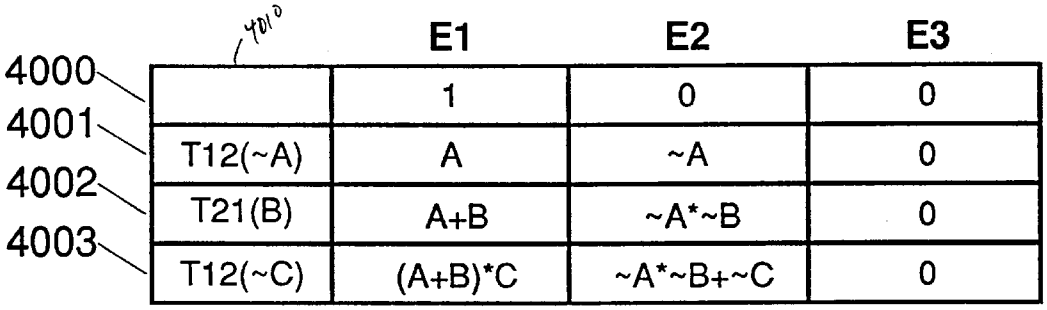

Given two states E_j and E_k, and a negated In-expression condition
3801
3810
Move slaves from E_j to E_k that do not satisfy first primitive condition
3800
3820
Repeat for each remaining primitive condition
DONE
3830
is the operator an AND or an OR ?
AND
OR
3831
Move slaves from E_k to E_j that satisfy primitive condition
3832
Move slaves from E_j to E_k that do not satisfy primitive condition
3802
Slaves that satisfy condition moved state E_k

FIG. 38



	E1	E2	E3
3900	1	0	0
3901 T12(A)	~A	A	0
3902 T12(B)	~A*~B	A+B	0
3903 T21(~C)	~A*~B+~C	(A+B)*C	0

FIG. 39



	E1	E2	E3
4000	1	0	0
4001 T12(~A)	A	~A	0
4002 T21(B)	A+B	~A*~B	0
4003 T12(~C)	(A+B)*C	~A*~B+~C	0

FIG. 40

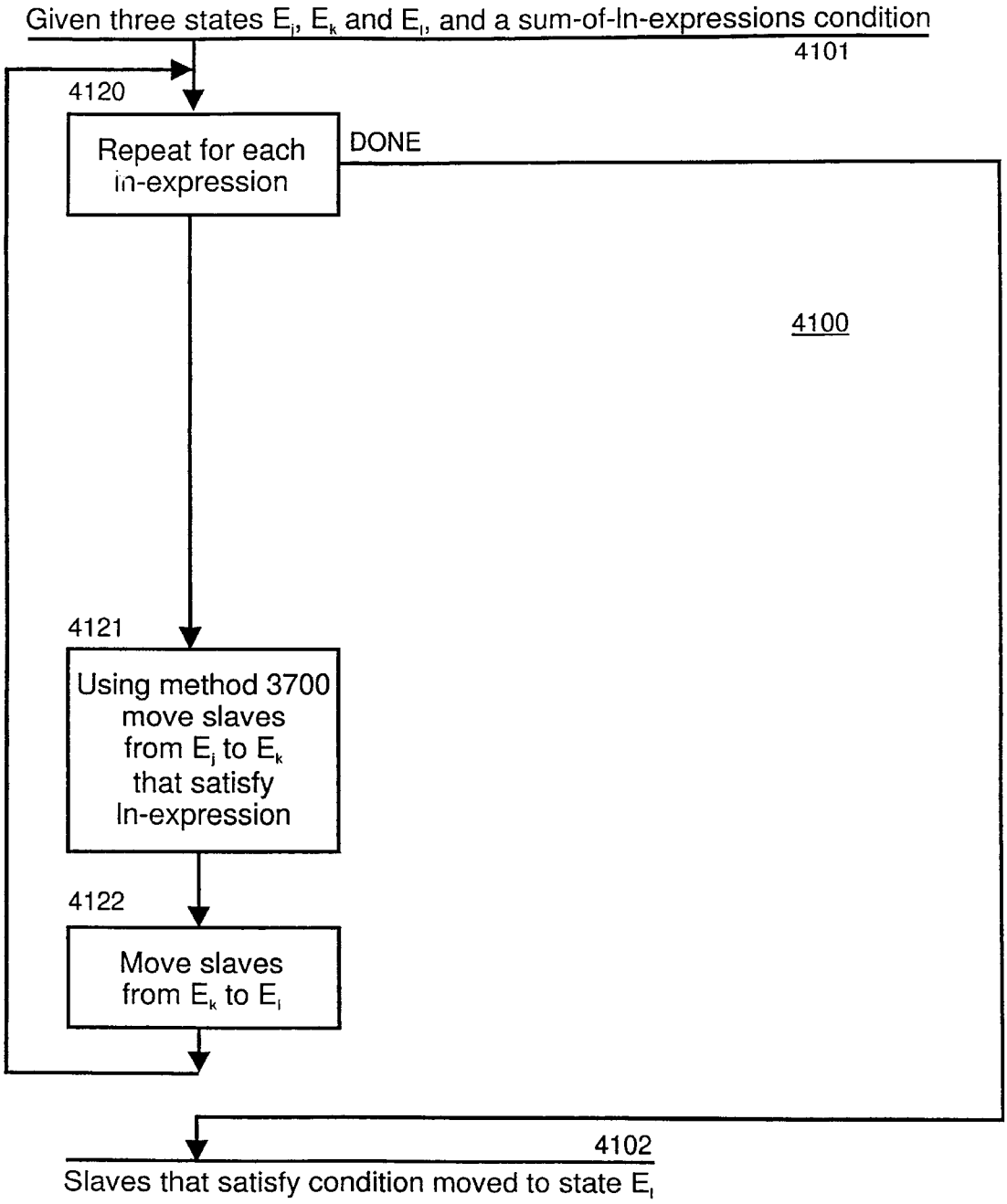

Given three states E_j, E_k and E_l, and a sum-of-ln-expressions condition
4101
4120
Repeat for each ln-expression
DONE
4100
4121
Using method 3700 move slaves from E_j to E_k that satisfy ln-expression
4122
Move slaves from E_k to E_l
4102
Slaves that satisfy condition moved to state E_l

FIG. 41

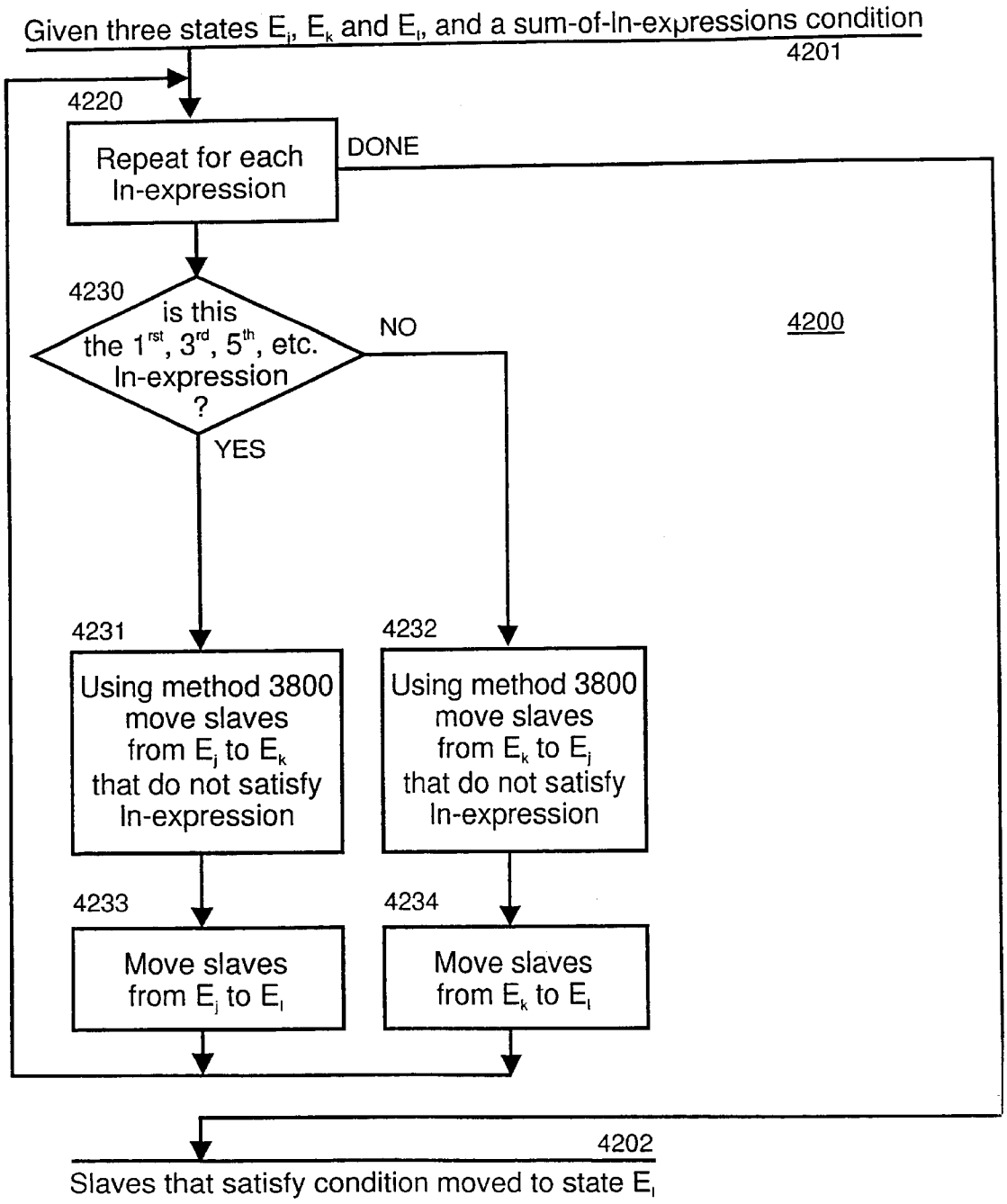

Given three states E_j, E_k and E_l, and a sum-of-ln-expressions condition
4201
4220
Repeat for each ln-expression
DONE
4230
is this the 1st, 3rd, 5th, etc. ln-expression ?
NO
YES
4200
4231
Using method 3800 move slaves from E_j to E_k that do not satisfy ln-expression
4232
Using method 3800 move slaves from E_k to E_j that do not satisfy ln-expression
4233
Move slaves from E_j to E_l
4234
Move slaves from E_k to E_l
4202
Slaves that satisfy condition moved to state E_l

FIG. 42

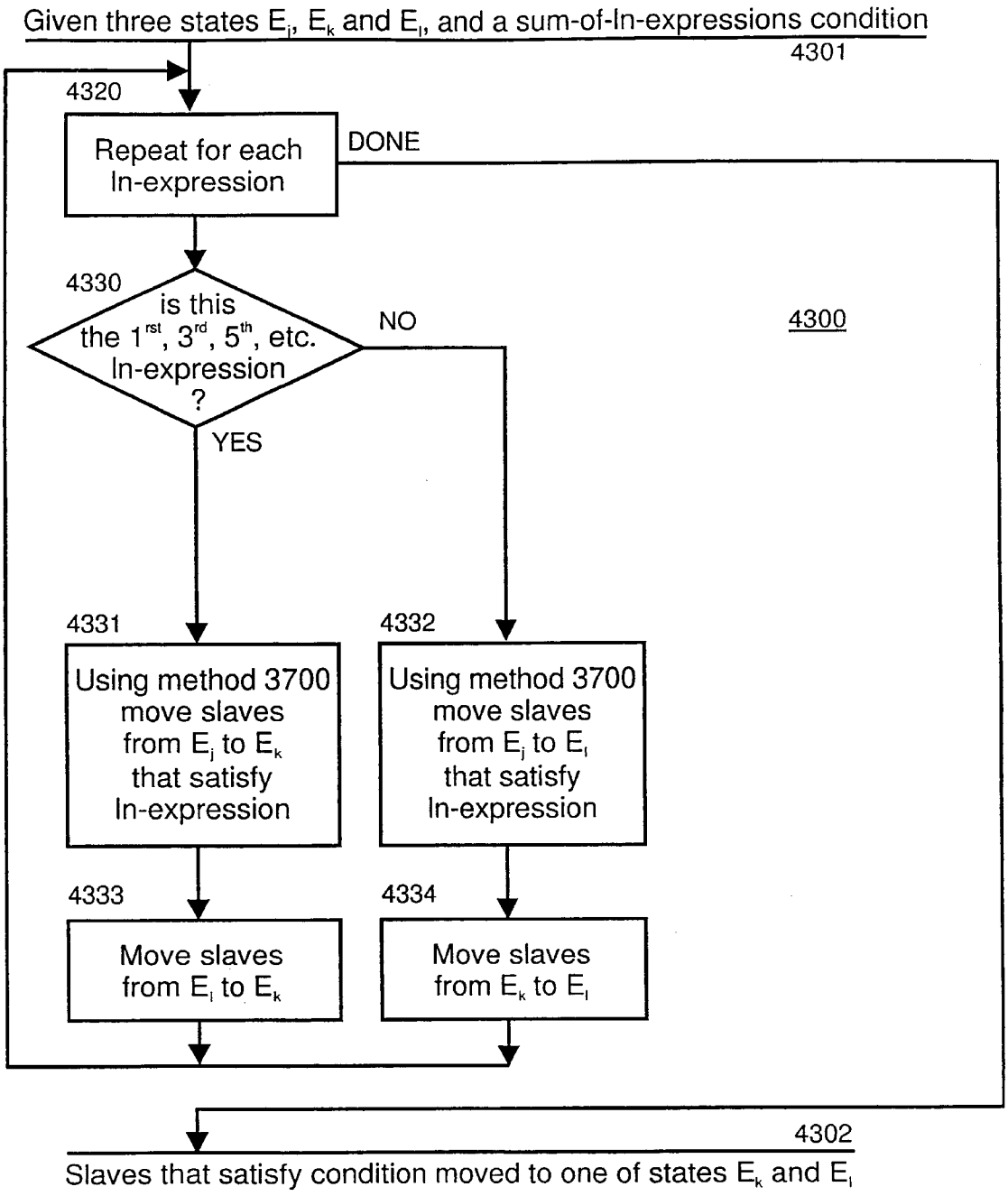

Given three states E_j, E_k and E_l, and a sum-of-In-expressions condition
4301
4320
Repeat for each In-expression
DONE
4330
is this the 1rst, 3rd, 5th, etc. In-expression ?
NO
4300
YES
4331
Using method 3700 move slaves from E_j to E_k that satisfy In-expression
4332
Using method 3700 move slaves from E_j to E_l that satisfy In-expression
4333
Move slaves from E_l to E_k
4334
Move slaves from E_k to E_l
4302
Slaves that satisfy condition moved to one of states E_k and E_l

FIG. 43

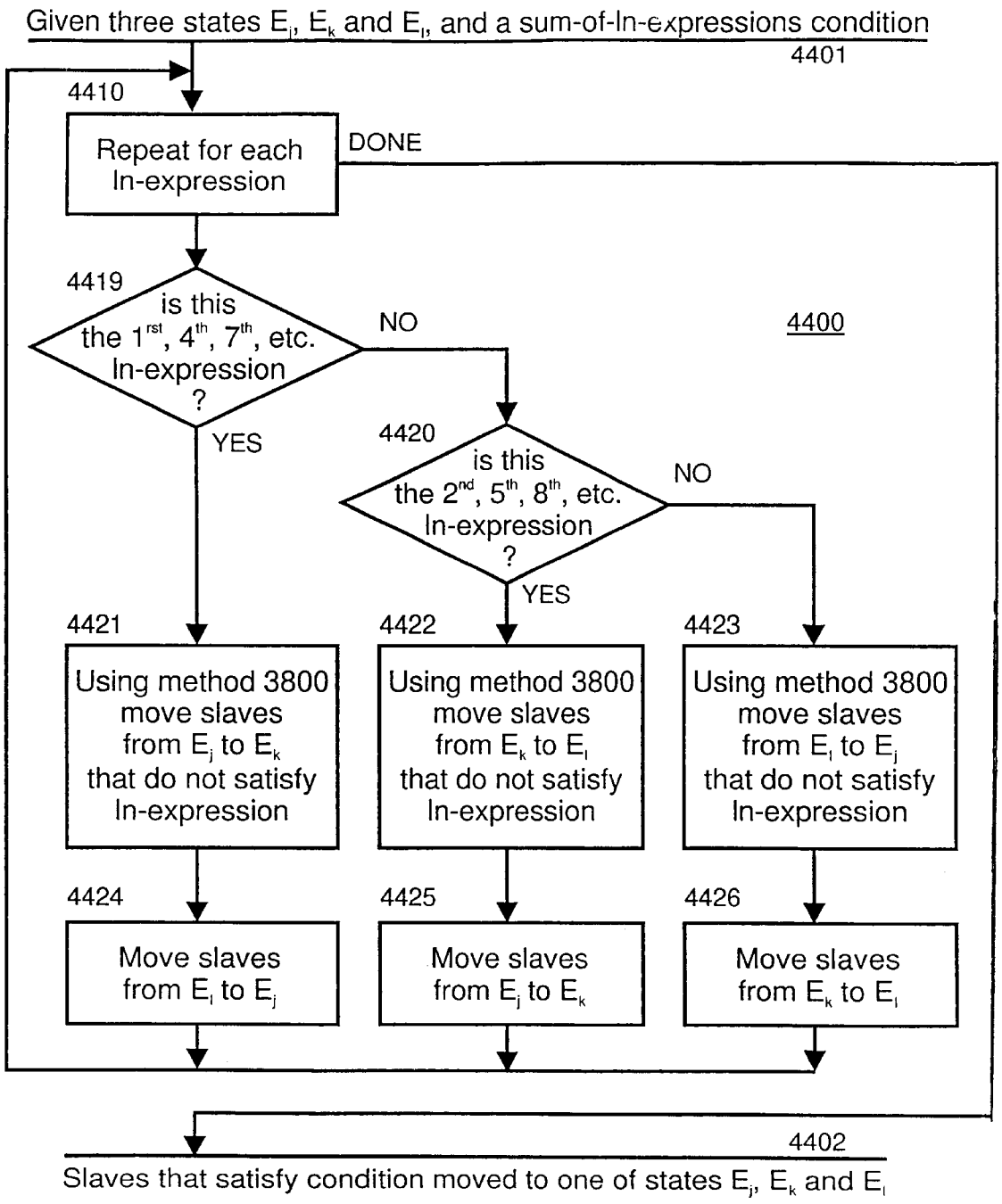

Given three states E_j, E_k and E_l, and a sum-of-ln-expressions condition
4401
4410
Repeat for each ln-expression
DONE
4419
is this the 1rst, 4th, 7th, etc. ln-expression ?
NO
4400
YES
4420
is this the 2nd, 5th, 8th, etc. ln-expression ?
NO
YES
4421
Using method 3800 move slaves from E_j to E_k that do not satisfy ln-expression
4422
Using method 3800 move slaves from E_k to E_l that do not satisfy ln-expression
4423
Using method 3800 move slaves from E_l to E_j that do not satisfy ln-expression
4424
Move slaves from E_l to E_j
4425
Move slaves from E_j to E_k
4426
Move slaves from E_k to E_l
4402
Slaves that satisfy condition moved to one of states E_j, E_k and E_l

FIG. 44

SYSTEM AND METHOD FOR SELECTING A SUBSET OF AUTONOMOUS AND INDEPENDENT SLAVE ENTITIES

CROSS REFERENCE TO RELATED APPLICATIONS

The present application is a continuation of application Ser. No. 08/646,539 filed May 8, 1996, now U.S. Pat. No. 5,828,318 issued Oct. 27, 1998. The present application is also a continuation-in-part of application Ser. No. 08/694,606 filed Aug. 9, 1996, now U.S. Pat. No. 5,942,987 issued Aug. 24, 1999, which in turn is a continuation-in-part of application Ser. No. 08/303,965 filed Sep. 9, 1994, now U.S. Pat. No. 5,673,037 issued Sep. 30, 1997.

FIELD OF THE INVENTION

The invention relates to communications between a master station and one or more slave stations. More specifically, the invention relates to a master station selecting subset(s) of the slave stations by broadcasting commands with conditions that the selected slaves meet.

BACKGROUND OF THE INVENTION

In the prior art, a master control unit is to communicate with a plurality of autonomous and independent slaves. In such environment, the number of slaves is often not known a priori. There may in fact be no slaves with which the master can communicate. Among the reasons the master may have to communicate with the slaves are (a) the need to acknowledge their presence, (b) identify and count them and/or (c) order them to perform tasks. This kind of computational environment falls under the broad category of broadcasting sequential processes, which is defined by Narain Gehani in Chapter 9 of the book co-edited with Andrew McGettrick, "Concurrent Programming" (Addison-Wesley, 1988), which is herein incorporated by reference in its entirety.

Because the master often does not know ahead of time the number of slaves present and because that number may be very large and possibly unyieldy, it is advantageous for the master to be able to select a subset of the slaves with whom to communicate further. Such a selection must of course be done by a conditional. Those slaves that meet the condition are thus considered selected, while those that do not meet the condition are considered not selected. The selection is performed by broadcasting to all slaves the condition that must be met. This is akin to asking those among a large crowd of people whose last name is Lowell to raise their hand. Each slave is defined has having at least the capability to listen to the master's broadcasts, to receive the broadcast condition and to self-test so as to determine whether it meets the condition. See for example, U.S. patent application Ser. No. 08/303,965, entitled "Radio Frequency (RF) Group Select Protocol" to Cesar et al. filed on Sep. 9, 1994 which is herein incorporated by reference in its entirety.

Practical environments where this computational model can be applied include bus arbitration, wireless communication, distributed and parallel processing. Characteristic of such environments is the existence of a protocol for how master and slaves communicate. The aforementioned capability of subset selection can be an important additional component of that protocol.

Finite state-machines are a well-known modelling tool. The set theory that often accompanies the definition of finite state-machines is also well known. Both subjects are amply covered in any of many books on discrete or finite mathematics that are available today. The book by Ralph Grimaldi, "Discrete and Combinatorial Mathematics: An Applied Introduction" (Addison-Wesley, 1985), is a fine example of its kind.

STATEMENT OF PROBLEMS WITH THE PRIOR ART

In the prior art, the methods used for selecting subsets of slaves is limited to comparisons against the information held by the slaves such that the comparison is either true or false and the slaves can be in either of two selection states: selected or not selected. The slave may contain many other states, but only two are effectively dedicated to the purpose of subset selection.

In some older prior art, a comparison will override a previous comparison. There is no notion of accumulating and combining successive comparisons to effect a selection according to a complex condition.

More recent prior art allows slaves to move between two selection states according to successive comparisons. That allows some complex conditions to be effected. However not all complex conditions can be effected with such two-selection-state machine. For example, the complex condition "is-red and is-not tall or is-not-red and is-tall", that is, the EXCLUSIVE-OR of the two simple comparisons "is-red" and "is-tall", can not be performed such that the subset of slaves that satisfy the EXCLUSIVE-OR are in the first state and those that do not satisfy the EXCLUSIVE-OR are in the second state. In the case of complex conditions involving two comparisons and their negation, the two-selection-state machine can not perform the EXCLUSIVE-OR and the EQUIVALENCE logical operators. In the case of complex conditions involving more than two comparisons and their negation, the two-selection-state machine can not perform an increasingly large number of logical equations.

In the prior art, conditions such as the EXCLUSIVE-OR must be broken up into two independent processing steps. First, slaves satisfying the first AND term are selected and all necessary processing sequence is performed over them. Second, after a general reset, slaves satisfying the second AND term are selected and the same necessary processing sequence is repeated over those. That means that the processing sequence must be broadcast twice. In the case of more complicated conditions, rebroadcasting of such sequence may happen more than twice. For example, the condition (A*~B*~C)+(~A*B*~C)+(~A*~B+C) would need three rebroadcasts.

The only conditions that can be executed in a single round of broadcasting by a two-selection-state logic as used by the prior art are those conditions that can be expressed by a left-nested expressions, such as ((A+B+C)*D*E)+F). OR conditions, such as (A+B+C+D), and AND conditions, such as (A*B*C), are particular cases of left-nested expressions. In contrast, EXCLUSIVE-OR type conditions, such as (A*B*~C)+(A*~B*C)*(~A*B*C), can not be written as left-nested expressions and therefore can not be handled by the two-selection-state logic.

Among the prior art is U.S. Pat. No. 5,434,572, entitled "System and Method for Initiating Communications between a Controller and a Selected Subset of Multiple Transponders in a Common RF Field" to Smith dated Jul. 18, 1995. The method begins by selecting all "transponders" in a field and, by a sequence of commands, incrementally moving groups of slaves to a "reset" condition.

In the U.S. Pat. No. 5,410,315, entitled "Group-Addressable Transponder Arrangement" to Huber dated Apr. 25, 1995, a selection is essentially made through a comparison against a "group and/or unit address". Unlike Smith, there is no progressive incremental refinement. Huber can perform only limited AND operations.

OBJECTS OF THE INVENTION

An object of this invention is a system and method for using arbitrarily complex logical conditions to select slave stations that satisfy those conditions transmitted by a master station through a series one or more commands.

An object of this invention is a system and method for using arbitrarily complex logical conditions to select RF transponders that satisfy those conditions transmitted by a base station through a series one or more commands.

SUMMARY OF THE INVENTION

The present invention is a system and method for selecting a subset of a plurality of autonomous and independent slaves, wherein each slave comprises (i) a three-state machine dedicated to selection, (ii) some other stored information, and (iii) a logic to execute externally provided commands in a command sequence that exercise the three-state machine. The primary purpose of the commands is to effect state transitions. The slave receives the command, which causes a comparison to be performed against the slave's stored information, the results of which possibly causing a state transition in the slave.

The commands, in a sequence called a command sequence, are broadcast from at least one master control unit to zero or more slaves. The exact number of slaves may not be known by the master. The master executes a method by which a sequence of discrete commands is broadcast to all slaves. The overall purpose of the method is to bring a subset of the slaves to be at the same state of their three-state machine, while all other slaves are at any one of the two other remaining states.

A three-state machine dedicated to selection is present in every slave. Each slave is at one of those three states, therefore, at any one time, the slaves can be subdivided into three subsets: those slaves that have their selection three-state machine at the first state, those at the second state, and those at the third state. In a preferred embodiment, transitions are possible between any two states of the three-state machine. Transitions are requested by command (sequence) broadcast from the master. A command specifies a desired transition, say from the second state to the first state. Only slaves that are at the second state may be affected. The command also specifies a condition under which the transition will occur. If the condition is met, the transition is effected; if not, the slave remains in its previous state.

In a preferred embodiment, slaves can be moved from a first state to a second state and visa versa. Only slaves in the second state can be moved to a third state. The slaves in the third state ignore the remaining commands in the command sequence. In alternative preferred embodiments, the first and second states reverse roles after an end of one or more subsequences in the sequences of commands. Also, the second and third states can reverse roles after an end of one or more subsequences. Further, the states of the slaves can cycle their roles at the end of one or more of the subsequences.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of preferred embodiments of the invention with reference to the drawings that are included:

FIG. **1** is a block diagram of a master control unit broadcasting commands to a plurality of slaves.

FIG. **2** is a block diagram of the components of a master control unit.

FIG. **3** is a block diagram of the components of a slave.

FIG. **4** shows a state diagram showing a three-state machine that allows all six possible transitions between any two different states.

FIG. **5** shows a state diagram showing a three-state machine that allows five possible transitions between any two different states.

FIG. **6** shows a state diagram showing a three-state machine that allows four possible transitions between any two different states, such that any state is reachable from any other state.

FIG. **7** shows a state diagram showing a three-state machine that allows four possible transitions between any two different states, such that any state is reachable from any other state and two of the states have no transitions between them.

FIG. **8** shows a state diagram showing a three-state machine that allows three possible transitions between any two different states, such that any state is reachable from any other state.

FIG. **9** lists all possible three-state machines that can be used for the purposes of this invention.

FIG. **10** is a set theoretic representation of how the plurality of slaves is subdivided into at most three sets.

FIG. **11** describes what happens when a command to transfer slaves that satisfy some condition from one state to another is broadcast.

FIGS. **12**, **13**, **14**, **15**, **16** and **17** exemplifies by means of Venn diagrams how a command transfers elements from one set to another.

FIGS. **18**, **19** and **20** show sequences of commands for selecting a subset of slaves that satisfy an EXCLUSIVE-OR condition and such that those slaves end up in a first, second and third set, respectively.

FIG. **21** describes a method that computes a product (AND) condition.

FIG. **22** describes a method that computes a negated product (NAND) condition.

FIG. **23** describes a method that computes a sum (OR) condition.

FIG. **24** describes a method that computes a negated sum (NOR) condition.

FIG. **25** describes a method that computes a condition written in sum-of-products form whereby the product terms are built in a second state and the sum is accumulated in a third state.

FIGS. **26** and **27** exemplify the use of the method described in FIG. **25**.

FIG. **28** describes a method that computes a condition written in sum-of-products form whereby the product terms are built alternatively in a first and second states and the sum is accumulated in a third state.

FIGS. **29** and **30** exemplify the use of the method described in FIG. **28**.

FIG. **31** describes a method that computes a condition written in sum-of-products form whereby the product terms are built alternatively in a second and third states and the sum is accumulated in that second and third state, respectively.

FIGS. **32** and **33** exemplify the use of the method described in FIG. **31**.

FIG. **34** describes a method that computes a condition written in sum-of-products form whereby the product terms are built alternatively in a first, second and third states and the sum is accumulated in that first, second and third state, respectively.

FIGS. **35** and **36** exemplify the use of the method described in FIG. **31**.

FIG. **37** describes a method that computes a single left-nested expression using only two states.

FIG. **38** describes a method that computes the negation of a single left-nested expression using only two states.

FIG. **39** exemplifies the use of the method described in FIG. **37**.

FIG. **40** exemplifies the use of the method described in FIG. **38**.

FIG. **41** describes a method that computes a condition written in sum-of-left-nested-expressions form whereby the left-nested expressions are built in a second state and the sum is accumulated in a third state.

FIG. **42** describes a method that computes a condition written in sum-of-left-nested-expressions form whereby the left-nested expressions are built alternatively in a first and second states and the sum is accumulated in a third state.

FIG. **43** describes a method that computes a condition written in sum-of-left-nested-expressions form whereby the left-nested expressions are built alternatively in a second and third states and the sum is accumulated in that second and third states, respectively.

FIG. **44** describes a method that computes a condition written in sum-of-left-nested-expressions form whereby the left-nested expressions are built in a first, second and third states and the sum is accumulated in that first, second and third states, respectively.

DETAILED DESCRIPTION OF THE INVENTION

FIG. **1** shows least one master control unit **101** communicating with a plurality of autonomous and independent slaves **102**. The communication medium **103** can be in terms of either direct electric contact means or electromagnetic radiation means, e.g., radio frequency (RF) embodiments. However, it also encompasses light, sound and other frequencies utilized for signalling purposes. Master unit **101** is capable of broadcasting commands at the plurality of slaves **102** via communication medium **103**. Each slave is capable of receiving the broadcast commands which are processed by logic **150**. For example, U.S. Pat. No. 4,656,463 to Anders et al. shows an RF tag systems where for our purposes the active transceiver (AT) would be the master control unit **101**, the passive transceivers (PT) would be the independent slaves **102**, and the communication medium **103** would be RF over free space. In an alternative preferred embodiment, U.S. Pat. No. 5,371,852 to Attanasio et al. shows a cluster of computers where for our purposes the gateway would be the master unit **101**, the nodes in the cluster would be the slaves **102**, and the communication medium **103** would be the interconnect. These references are incorporated by the reference in their entirety.

FIG. **2** shows that a master unit **200** comprises (a) means **201** for broadcasting commands from a command set **250** to a plurality of slaves, and (b) processing means **202** for determining the correct sequence of commands to be broadcast. In one embodiment, processing means **202** are proximate to broadcast means **201**. Other embodiments are possible where processing means **202** and broadcast means **201** are remote from each other.

FIG. **3** shows that a slave **300** comprises (a) a receiver **301** for receiving commands from a master, (b) a three-state machine **304**, (c) a memory with stored information **303** (d) processor or logic **302** for executing any received command, performing any condition testing over the stored information **303** as specified by the command, and effecting a state transition on three-state machine **304** conditional to the result of the condition testing. Receiving means **301** and broadcasting means **201** must by necessity be compatible. The receiving means **301** are well known. For example in RF tagging, radio frequency receivers/transmitters are used. In the networking arts standard connections to networks and/or information buses are used. Stored information **303** is typically embodied in the form of registers and/or memory.

The three-state machine **304** comprises a select state, a first unselect state, and a second unselect state state. All three of these states can be used in the selection and unselection of sets of slaves. The logic **302** is further described below.

The significant difference between a two and a three-selection-state machine is that, using any sequence of commands, the former can only isolate or select slaves that satisfy a condition expressed by a left-nested expression. Using a two-selection-state machine, there is no sequence of commands that can process conditions that are expressed as as a SUM-of-left-nested-expressions.

Only a three-selection-state machine can select slaves that satisfy a condition expressed by a sum-of-left-nested-expression. Further, a three-selection-state machine is also sufficient to select a set of slaves that satisfy any arbitrary condition, even though those conditions are expressed by a sum-of-left-nested-expression. Therefore adding a fourth, fifth, etc. state does not add any new capability to the selection logic. In addition, since any condition can be expressed by a sum-of-left-nested expression, a three-selection-state machine can select a set of slave satisfying any possible condition. This capability is undisclosed and unrecognized in the prior art.

The invention enables this capability because a separate condition (or set of conditions), each corresponding to a set of slaves, can be isolated in any one of the three states at any given time. Therefore, operations on two sets of conditions, in two of the respective states, can be performed without affecting or being affected by the conditions held in the third state.

In one embodiment, receiving means **301**, processing means **302**, stored information **303**, and three-state machine **304** are proximate. Other embodiments are possible where the components **301**, **302**, **303** and **304** are remote from each other, in part or completely.

The three states are dedicated to the process of determining whether a slave satisfies an arbitrarily complex condition. During the process of determining whether the slave satisfies the condition, the slave may be in any of the three states as dictated by the process. If the slave does satisfy the condition, the process assures that the slave will end up at a state that enables the slave to communicate further with the master.

Three-state machine **304** is part of every slave. A preferred three-state machine is shown in FIG. **4**. Three-state machine **400** includes the three states **401**, **402** and **403**, and all six possible state-to-state transitions **412**, **413**, **421**, **423**, **431** and **432**, where the transitions are between two different states. The three states **401**, **402** and **403** are named S1, S2

and S3, respectively. A transition from a state to itself is not helpful for the methodology described herein.

Other three-state machines are possible. In FIG. **5**, three-state machine **500** has only five of the six state-to-state transitions present in the three-state machine **400**. Three-state machine **500** represents a class of six possible three-state machines where one of the six possible state-to-state transitions is inoperative. In the particular case of FIG. **5**, transition **513** is inoperative.

In FIG. **6**, three-state machine **600** has only four of the six state-to-state transitions of three-state machine **400**. The four operative transitions are such that any state can be reached from another state by means of one or two transitions. Moreover there is at least one possible transition between any two states. Three-state machine **600** represents a class of six possible three-state machines were two of the six possible state-to-state transitions are inoperative, while still providing access to any state from any other state. In the particular case of FIG. **6**, transitions **413** and **432** are inoperative.

In FIG. **7**, three-state machine **700** has only four of the six state-to-state transitions of three-state machine **400**. The four operative transitions are such that any state can be reached from another state by means of one or two transitions. Moreover there are two states between which there is no possible transition. Three-state machine **700** represents a class of three possible three-state machines where one of the pairs of transitions between two states is inoperative. In the particular case of FIG. **7**, the pair of transitions **413** and **431** between states **401** and **403** is inoperative.

In FIG. **8**, three-state machine **800** has only three of the six state-to-state transitions of three-state machine **400**. The three operative transitions are such that any state can be reached from another state by means of one or two transitions. Therefore each pair of states is connected by one and only one transition, in such a way that all transitions move either clockwise or anticlockwise. Three-state machine **800** represents a class of two possible three-state machines that have only three of the six state-to-state transitions. In the particular case of FIG. **8**, the three operative transitions **412**, **423** and **431** define an anticlockwise cycle.

All three-state machines relevant to this invention are listed in FIG. **9**. The six state-to-state transitions **412**, **421**, **423**, **432**, **431** and **413** are named T**12**, T**21**, T**23**, T**32**, T**31** and T**13**, respectively. Each row defines one possible three-state machine. For each state-to-state transition, the existence or not of that transition is indicated. The three-state machine defined by row **900** is a preferred embodiment that corresponds to three-state machine **400** of FIG. **4**. The three-state machines defined by rows **901**, **902**, **903**, **904**, **905**, and **906** belong to the class of three-state machine **500** of FIG. **5**. The three-state machines defined by rows **907**, **908**, **909**, **910**, **911**, and **912** belong to the class of three-state machine **600** of FIG. **6**. The three-state machines defined by rows **913**, **914**, and **915** belong to the class of three-state machine **700** of FIG. **7**. The three-state machines defined by rows **916** and **917** belong to the class of three-state machine **800** of FIG. **8**.

At any one time, each slave is at one and only one of the three states **401**, **402**, or **403**. Accordingly, there are three sets of slaves: those are state **401**, those are state **402**, and those at state **403**. State transitions are equivalent to movement between those three sets. This view of operating over sets is illustrated in FIG. **10**. Set **1001**, named E**1**, contains as elements all slaves **1010** that are at state **401**. Set **1002**, named E**2**, contains as elements all slaves **1020** that are at state **402**. Set **1003**, named E**3**, contains as elements all slaves **1030** that are at state **403**. There are six base commands, irrespective of the condition they specify, for effecting movement of elements between those three sets. Command **1012**, named T**12**, moves elements from set **1001** to set **1002**. Command **1021**, named T**21**, moves elements from set **1002** to set **1001**. Command **1023**, named T**23**, moves elements from set **1002** to set **1003**. Command **1032**, named T**32**, moves elements from set **1003** to set **1002**. Command **1031**, named T**31**, moves elements from set **1003** to set **1001**. Command **1013**, named T**13**, moves elements from set **1001** to set **1003**.

The simplest form of command is illustrated in FIG. **11**. Command **1110** comprises three parameters. First, the "from" state **1101**; second, the "to" state **1102**; and third, the primitive condition **1112** which must be satisfied for the transition to happen. Those three parameters are named Si, Sj, and s, respectively in the figure. Si, the "from" state **1101**, and Sj, the "to" state **1102**, can be any of the three states **401**, **402** or **403**, except that Si and Sj are not equal.

The primitive condition may take many forms depending on the overall capabilities of the slaves and the purposes that underlie the need for selecting subsets of slaves. Such primitive conditions could take the form of equality testing or numerical comparisons. Even though a single command broadcast from the master to the slave can only specify a single primitive condition, arbitrarily complex conditions are realized by a sequence of these primitive commands. In a preferred embodiment, an arbitrarily complex condition is described by a logical equation over primitive conditions. For example, the complex condition A*B+~A*~B, where A and B are primitive conditions, "*" is the binary logical operator AND, "+" the binary logical operator OR, and "~" the unary logical operator NOT. Negated primitive conditions, such as ~A, are assumed to be primitive conditions.

It is convenient for expositional purposes to textually represent command **1110**. A simple syntax used herein is to write the command as Tij(s), where i and j are 1, 2, or 3, corresponding to states **401**, **402** and **403**, respectively, and s is the condition to be satisfied. The prefix T, for transition, is purely cosmetic. For example, T**31**(~A) represents a command to move all slaves that are at the third state and which do not satisfy A, to the first state, while T**23**(**1**) represents a command to move all slaves that are at the second state to the third state unconditionally.

The six possible transitions **412**, **413**, **421**, **423**, **431** and **432** for some condition s, can thus be written has T**12**(s), T**13**(s), T**21**(s), T**23**(s), T**31**(s) and T**32**(s), respectively. Any command thus involves only two of the three states of a three-state machine and only one of the six possible transitions. The pair of states **1100** in FIG. **11** and the single transition between them defines the scope of a single command. Only slaves that are at state Si, that is the "from" state **1101**, of the pair of states **1100** are allowed to transition, and those that do transition will do it to state Sj, that is the "to" state **1102**. Condition s is tested by each slave that is at state Si. Those for which the condition is satisfied will switch to state Sj. These semantics are expressed by the two logical expressions

$$Ei=Ei*\sim s$$

$$Ej=Ej+Ei*s$$

The first expression states that the set Ei, of all slaves that are at state Si, is decremented by the number of slaves that

move from Si to Sj. That is expressed in the form of a logical AND between the previous value of set Ei and the virtual set of all slaves in Ei that did not satisfy condition s. Concurrently, the second expression states that are the set Ej, of all slaves that are at state Sj, is augmented by the number of slaves that have moved from Si to Sj. That is expressed in the form of a logical OR between the previous value of set Ej and the virtual set of all slaves in Ei that satisfy condition s, the latter expressed by a logical AND between the previous value of set Ei and the virtual set of all slaves in Ei that satisfy condition s.

Since a command **1110** is broadcast to all slaves and they receive and operate on it concurrently, the command is essentially an operation over sets. The command Tij(s) effectively moves elements from a set Ei, of all slaves at state Si, to a set Ej, of all slaves at state Sj. Therefore sets E**1**, E**2** and E**3** are associated to states S**1**, S**2** and S**3**, respectively. Those two notions, sets and states, are for the purpose of this invention functionally equivalent. Reference herein to states S**1**, S**2** and S**3** imply sets E**1**, E**2** and E**3**, and vice versa, respectively.

FIG. **12** illustrates by means of Venn diagrams a simple non limiting example of one of such command. Sets **1001**, **1002** and **1003** correspond to slaves that are in the first, second and third state, respectively. For this example, set **1001** initially contains all slaves, while sets **1002** and **1003** are empty. Three testable primitive conditions A, B and C are defined. Each one of these three primitive conditions defines a virtual subset **1205**, **1206** and **1207**, respectively. The negation of each primitive condition, namely ~A, ~B and ~C, defines three complementary virtual subsets to **1205**, **1206** and **1207**, respectively. Those virtual subsets may or not have slaves in common. In the figure we assume the most difficult case where virtual subsets **1205**, **1206** and **1207** intersect each other. Command T**12**(A) is broadcast, which forces all slaves in virtual subset **1205** in set **1001** to move from set **1001** to set **1002**. Therefore, after the command is executed by all slaves in this situation, the right half of the figure shows that set **1001** represents the ~A condition, set **1002**, the A condition, and set **1003** is empty.

Note that other Figures in this disclosure that are Venn diagrams have there sets numbered in the same manner as FIG. **12** but that these number are not shown for clarity.

Another example is shown in FIG. **13**. The starting configuration is the same as in FIG. **12**. However, the command T**12**(~A) is broadcast instead. As the right half of the figure indicates, after the command is executed by all slaves in this situation, set **1001** represents the A condition, set **1002**, the ~A condition, and set **1003** is empty.

A condition that is the product of two or more primitive conditions is obtained by two or more commands. An AND condition A*B, for example, can be obtained by broadcasting two commands. First, T**12**(A), then T**21**(~B). Starting from the same initial configuration as used in FIG. **12**, execution of T**12**(A) is shown in FIG. **12**. Execution of command T**21**(~B) on that resulting configuration is shown in FIG. **14**. As the right half of the figure indicates, after the two commands are executed in succession by all slaves in this situation, set **1001** represents the ~(A*B) condition, set **1002**, the A*B condition, and set **1003** is empty. Therefore, set **1002** represents an AND condition, while set **1001** represents the complementary NAND condition.

A condition that is the sum of two or more primitive conditions is obtained by two or more commands. An OR condition A+B, for example, can be obtained by broadcasting two commands. First, T**12**(A), then T**12**(B). Starting from the same initial configuration as used in FIG. **12**, execution of T**12**(A) is shown in FIG. **12**. Execution of command T**12**(B) on that resulting configuration is shown in FIG. **15**. As the figure indicates, after the two commands are executed in succession by all slaves in this situation, set **1001** represents the ~(A+B) condition, set **1002**, the A+B condition, and set **1003** is empty. Therefore, set **1002** represents an OR condition, while set **1001** represents the complementary NOR condition.

The previous two examples involve only two of the possible three sets. Some conditions require the use of all three sets. The only two conditions involving two primitive conditions A and B that require all three sets are the EXCLUSIVE-OR, A*~B+~A*B, and its complement the EQUIVALENCE, A*B+~A*~B. (See discussion of left nested expressions below.) The EXCLUSIVE-OR can be obtained by broadcasting a sequence of three commands: T**12**(~A), T**13**(B) and T**21**(B). Execution of the three commands is shown in FIGS. **13**, **16**, and **17**. This sequence is more compactly represented in tabular form as done in FIG. **18**. Row **1800** of the table is the initial state of the three sets. In this example, set **1001** has all slaves and sets **1002** and **1003** are empty. Rows **1801**, **1802** and **1803** show the result of executing commands T**12**(~A), T**13**(B) and T**21**(B), respectively, and the resulting conditions expressed by each of the three sets after each command is executed. As FIG. **18** shows, after the three commands are executed in succession by all slaves in this situation, set **1001** represents the A*~B+~A*B condition, set **1002**, the ~A*~B condition, and set **1003**, the A*B condition. As is, the slaves that satisfy the EXCLUSIVE-OR condition ended up in set **1001**. If the goal had been to place those slaves in set **1002** instead, a different sequence of commands would be broadcast. That sequence is shown in FIG. **19**. Rows **1900**, **1901**, **1902** and **1903** of the table show the initial and succeeding conditions obtained in each set during the execution of the sequence of commands T**12**(A), T**23**(B) and T**12**(B). if the goal had been to place the EXCLUSIVE-OR in set **1003** instead, a different sequence of commands would be broadcast. That sequence is shown in FIG. **20**. Rows **2000**, **2001**, **2002** and **2003** of the table show the initial and succeeding conditions obtained in each set during the execution of the sequence of commands T**13**(A), T**32**(B) and T**13**(B).

The present invention teaches several methods for generating a command sequence necessary to select slaves that satisfy an arbitrarily complex condition involving any number of primitive conditions. Before describing the most general methods, the invention first teaches a few important methods aimed at certain types of complex conditions. Non limiting examples of command sequences are given in the columns numbered **1810** (in FIG. **18**), **1910** (in FIG. **19**), **2010** (in FIG. **20**), **2610** (in FIG. **26**), **2710** (in FIG. **27**), **2910** (in FIG. **29**), **3010** (in FIG. **30**), **3210** (in FIG. **32**), **3310** (in FIG. **33**), **3510** (in FIG. **35**), **3610** (in FIG. **36**), **3910** (in FIG. **39**), and **4010** (in FIG. **40**).

Each command, T, sent from the master to one or more slaves, has a single primitive condition, ci, that is one of any number of arbitrary primitive conditions. The command, T, addresses some information stored on each of the slaves and causes the respective slave to compare its stored information with the primitive condition.

The first of those is a condition expressed by the product of two or more primitive conditions. This is the general AND condition and the method for handling that kind of condition is shown in FIG. **21**. Method **2100** takes an input **2101** two different sets Ej and Ek of the possible three sets E**1**, E**2** and E**3**, and an AND of N primitive conditions, that is, c**1***c**2*** . . . *cN. Method **2100** outputs a configuration **2102** whereby

all slaves in set Ej that satisfied the aforementioned AND condition have moved to set Ek. If set Ek was not empty to start with, a side effect of the method is that all slaves originally in Ek that did not satisfy the reduced condition c2* . . . *cN have moved to set Ej. That can be represented mathematically as

Ej=Ej*~(c1*c2* . . . *cN)+Ek*~(c2*c3* . . . *cN)

Ek=Ej*(c1*c2* . . .*cN)+Ek*(c2*c3* . . . *cN)

Method **2100** accomplishes this by generating a sequence of N commands. In step **2110**, a command is issued that causes all slaves in set Ej that satisfy first primitive condition c1 to move to set Ek. This command is written as Tjk(c1). The state transitions effected can be mathematically represented as

Ej=Ej*~c1

Ek=Ek+Ej*c1

Step **2120** controls the iteration over all remaining primitive conditions that make up the input AND condition **2101**. For each primitive condition ci, where i varies from 2 to N, step **2121** issues a command that causes all slaves in set Ek that do not satisfy primitive condition ci to move to set Ej. This command is written as Tkj(~ci). The state transitions effected can be mathematically represented as

Ej=Ej+Ek*~c1

Ek=Ek*c1

The iteration ends after the last primitive condition, cN, has been processed by step **2121**. That terminates the method.

In a preferred embodiment, Ek begins as a null set so that the slaves that are found in Ek at the end of method **2100** are exactly those that satisfy the AND condition.

The second is a condition expressed by the negation of a product of two or more primitive conditions. This is the general NAND condition and the method for handling that kind of condition is shown in FIG. **22**. Method **2200** takes as input **2201** two different sets Ej and Ek of the possible three sets E1, E2 and E3, and a NAND of N primitive conditions, that is, ~(c1*c2* . . . *cN). Method **2200** outputs a configuration **2202** whereby all slaves in set Ej that satisfy the aforementioned NAND condition are moved to set Ek. That can be represented mathematically as

Ej=Ej*(c1*c2* . . . *cN)

Ek=Ek+Ej*~(c1*c2* . . . *cN)

The method **2200** accomplishes this by generating a sequence of N commands. The main step **2220** controls the iteration over all the primitive conditions that make up the input NAND condition. For each primitive condition ci, where i varies from 1 to N, step **2221** issues a command that causes all slaves in set Ej that do not satisfy primitive condition ci to move to set Ek. This command is written as Tjk(~ci). The state transitions effected can be mathematically represented as

Ej=Ej*c1

Ek=Ek+Ej*~c1

The iteration ends after the last primitive condition, cN, has been processed by step **2221**. That terminates the method.

In a preferred embodiment, Ek begins as a null set so that the slaves that are found in Ek at the end of method **2200** are exactly those that satisfy the NAND condition.

The third is a condition expressed by the sum of two or more primitive conditions. This is the general OR condition and the method for handling that kind of condition is shown in FIG. **23**. Method **2300** takes as input **2301** two different sets Ej and Ek of the possible three sets E1, E2 and E3, and an OR of N primitive conditions, that is, c1+c2+ . . . +cN. Method **2300** outputs a configuration **2302** whereby all slaves in set Ej that satisfy the aforementioned OR condition are moved to set Ek. That can be represented mathematically as

Ej=Ej*~(c1+c2+ . . . +cN)

Ek=Ek+Ej*(c1+c2+ . . . +cN)

Method **2300** accomplishes this by generating a sequence of N commands. The main step **2320** controls the iteration over all the primitive conditions that make up the input OR condition. For each primitive condition ci, wherein i varies from 1 to N, step **2321** issues a command that causes all slaves in set Ej that satisfy primitive condition ci to move to set Ek. This command is written as Tjk(ci). The state transitions effected can be mathematically represented as

Ej=Ej*~c1

Ek=Ek+Ej*c1

The iteration ends after the last primitive condition, cN, has been processed by step **2321**. That terminates the method.

In a preferred embodiment, Ek begins as a null set so that the slaves that are found in Ek at the end of method **2300** are exactly those that satisfy the OR condition.

The fourth is a condition expressed by the negation of the sum of two or more primitive conditions. This is the general NOR condition and the method for handling that kind of condition is shown in FIG. **24**. Method **2400** takes as input **2401** two different sets Ej and Ek of the possible three sets E1, E2 and E3, and a NOR of N primitive conditions, that is, ~(c1+c2+ . . . +cN). Method **2400** outputs a configuration **2402** whereby all slaves in set Ej that satisfy the aforementioned NOR condition are moved to set Ek. If set Ek was not empty to start with, a side effect of the method is that all slaves originally in Ek that satisfied the reduction condition c2+ . . . +cN have moved to set Ej. That can be represented mathematically as

Ej=Ej*(c1+c2+ . . . +cN)+Ek*(c2+c3+ . . . +cN)

Ek=Ej*~(c1+c2+ . . . +cN)+Ek*~(c2+c3+ . . . +cN)

Method **2400** accomplishes this by generating a sequence of N commands. In step **2410**, a command is issued that causes all slaves in set Ej that do not satisfy first primitive condition c1 to move to set Ek. This command is written as Tjk(~c1). The state transitions effected can be mathematically represented as

Ej=Ej*c1

Ek=Ek+Ej*~c1

Step **2420** then controls the iteration over all the remaining primitive conditions that make up the input NOR condition. For each primitive condition ci, where i varies from 2 to N, step **2421** issues a command that causes all slaves in set Ek that satisfy primitive condition ci to move to set Ej. This command is written as Tkj(ci).

Ej=Ej+Ek*ci

Ek=Ek*~ci

The iterative step **2420** ends after the last primitive condition, cN, has been processed by step **2421**. That terminates the method.

In a preferred embodiment, Ek begins as a null set so that the slaves that are found in Ek at the end of method **2400** are exactly those that satisfy the NOR condition.

Methods **2100**, **2200**, **2300** and **2400** can be combined to handle arbitrarily complex conditions. The simplest such combination is called canonical, because it is based on the well-known technique of expressing an arbitrarily complex condition in the form of sum-of-products, i.e. one or more ANDed primitive conditions that are ORed together. The canonical method works by computing each product term of the condition using two sets and accumulating, that is summing, the product terms into a third set.

FIG. **25** shows a method **2500** that moves slaves from the first to the second state and visa versa. When slaves are in the second state, it is possible to move them to the third state where all remaining commands in the command sequence are ignored. Method **2500** takes as input **2501** three different sets Ej, Ek, and El, that is, some permutation of sets E**1**, E**2** and E**3**, and a condition that is a sum of P product terms, that is, p**1**+p**2**+ . . . +pP. Assuming, for simplicity, that set Ek is empty at the start of the method. Method **2500** outputs a configuration **2502** whereby all slaves in set Ej that satisfy the aforementioned sum-of-products condition are moved to set El. That can be represented mathematically as

Ej=Ej*~(p**1**+p**2**+ . . . +pP)

Ek=0

El=El+Ej*(p**1**+p**2**+ . . . +pP)

Method **2500** accomplishes this by generating a sequence of commands. The main step **2520** controls the iteration over all the product terms that make up the input sum-of-products condition **2501**. For each product term pi, where i varies from 1 to P, first step **2521** issues a sequence of commands as defined by method **2100** that causes all slaves in set Ej that satisfy the product term pi to move to set Ek. Second step **2522** issues a command that causes all slaves in set Ek to move to set El. The iteration ends after the last product term, pP, has been processed by steps **2521** and **2522**. That terminates the method.

In other words, the method **2500** creates each of the product terms in **2521** and in set Ek. After each product term, pi, is created, it is ORed with the previously accumulated product terms in set El. That frees up set Ek in preparation for the next product term.

Applying the method **2500** to the EXCLUSIVE-OR condition ~A*B+A*~B, for example, results in the command sequence **2610** shown in FIG. **26**. The default initial configuration **2600** where all the slaves are in set E**1** is used. In this example, j=1, k=2 and l=3. Commands T**12**(~A) and T**21**(~B), as per method **2100**, are used to move slaves that satisfy the product term ~A*B from set E**1** to set E**2**, as shown by rows **2601** and **2602**. Command T**23**(l) sums that product term from E**2** to E**3**, as shown in row **2603**. That is the basic cycle for one product term. Next, commands T**12**(A), T**21**(B) and T**23**(l) repeat the cycle to move slaves that satisfy the next product term A*~B first from set E**1** to set E**2** and second from set E**2** to set E**3**, as shown by rows **2604**, **2605** and **2606**. Six commands are generated by method **2500** for the EXCLUSIVE-OR condition, two of those on account of two product terms and the other four on account of four primitive conditions present over all product terms. Contrast this with the three commands generated by any the hand crafted solution of FIGS. **18**, **19** and **20**. While the canonical method is easy to compute, the command sequences it generates are not minimal in general.

An arbitrarily complex condition can be written in the form of a sum-of-products. A canonical method for generating a command sequence for a sum-of-products is to use first and second states to calculate product terms and to use the third state to accumulate the product terms. The canonical method does not yield the shortest command sequence but is easy to compute.

When a complex condition is put in sum-of-product form, the products do not have to be expanded so that each contains all primitive conditions used in the complex condition. For an example which includes three primitive conditions A, B, and C, the complex condition A*~B*C+A*B*C+~A*B*C+A*B*C can be minimized to A*C+B*C and still be considered a sum-of-products for the purpose of method **2500** and other methods described hereunder. Such a minimization represents a significant reduction in commands required. While the fully expanded condition above would require sixteen commands (four product terms and twelve primitive condition appearances), the corresponding minimized sum-of-products requires six commands (two product terms and four primitive condition appearances), when both the command sequences are generated through the canonical method **2500**. The six commands solution **2710** is shown in FIG. **27** and not surprisingly mimics the command sequence **2610** of FIG. **26**. Again a default initial configuration **2700** where all slaves are in set E**1** is used. In this example, j=1, k=2 and l=3. Command T**12**(A) transfers from set E**1** to set E**2** the slaves in set E**1** that satisfy primitive condition A. Row **2701** indicates that set E**1** contains the slaves that satisfy condition ~A; set E**2**, condition A; and set E**3** is empty. Command T**21**(~C) transfers from set E**2** to set E**1** the slaves in set E**2** that do not satisfy primitive condition C. Row **2702** indicates that set E**1** represents condition ~A+A*~C, which is equivalent to ~A+~C; set E**2**, condition A*C; and set E**3** is empty. Command T**23**(l) transfers from set E**2** to set E**3** all slaves in set E**2**. Row **2703** indicates the accumulation of product term A*C into set E**3**. Command T**12**(B) transfers from set E**1** to set E**2** the slaves in set E**1** that satisfy primitive condition B. Row **2704** indicates that set E**1** represents the condition (~A+~C)*~B, which is equivalent to condition ~A*~B+~B*~C; and set E**2**, condition (~A+~C)*B, which is equivalent to ~A*B+B*~C. Command T**21**(~C) transfers from set E**2** to set E**1** the slaves in set E**2** that do not satisfy primitive condition C. Row **2705** indicates that set E**1** represents condition ~A*~B+~B*~C+(~A*B+B*~C)*~C, which reduces to ~A*~B+~B*~C+~A*B*~C+B*~C, then to ~A*~B+~B*~C+B*~C, then to ~A*~B+~C; and set E**2** represents condition (~A*B+B*~C)*C, which is equivalent to ~A*B*C. Command T**23**(l) transfers from set E**2** to set E**3** all slaves in set E**2**. Row **2706** indicates that set E**3** represents the condition A*C+~A*B*C, which reduces to (A+~A*B)*C, then to (A+B)*C, then to A*C+B*C, which is the sum-of-product condition that needed to be satisfied. Applying method **2500** to a minimized sum-of-products, as in this example, will generate a command sequence that is certainly shorter than a fully-expanded sum-of-products, but is still not necessarily minimal.

Characteristic of method **2500** is that the first set Ej serves as the main repository of slaves and will end up containing the slaves originally in set Ej that do not satisfy the sum-of-products condition. Second set Ek serves to build each product term. Third set El serves to sum the product terms and will end up containing the slaves originally in set Ej that

satisfy the sum-of-products condition. Therefore, for method **2500**, each of the three sets has a uniquely defined role.

This does not need to be the case and one can create variations of method **2500** where the roles alternate. The method shown in FIG. **28** is one such variation. Method **2800** differs from method **2500** in that the roles of states Ej (state **1**) and Ek (state **2**) alternate, i.e. states **1** and **2** reverse roles. Product terms are build alternatively in states Ek and Ej: first, in Ek, then in Ej, then back in Ek, and so on; in other words, odd product terms—first, third, fifth, etc.—are built in set Ek, while even product terms—second, fourth, sixth, etc.—are built in set Ej. Method **2800** is similar to method **2500** in that the role of state El remains the same. Method **2800** takes as input **2801** the same input as does method **2500**. Method **2800** outputs a configuration whereby all slaves in Ej that satisfy the sum-of-products condition are moved to set El, and either set Ej or Ek will be empty depending on the number of product terms. If the condition has an even number of product terms, Ek will be empty; otherwise, Ej will be empty. That can be represented mathematically as

El=El+Ej*(p**1**+p**2**+ . . . +pP)

(if P is even) Ej=Ej*~(p**1**+p**2**+ . . . +pP)

(if P is odd) Ej=0

(if P is even) Ek=0

(if P is odd) Ek=Ej*~(p**1**+p**2**+ . . . +pP)

The main step **2820** controls the iteration over all the product terms that make up the sum-of products condition **2801**. For each product term pi, where i varies from 1 to P, step **2830** tests whether i is odd or even. If i is odd, steps **2831** and **2833** are executed for product term pi. If i is even, steps **2832** and **2834** are executed for product term pi. Step **2831** issues a sequences of commands defined by NAND method **2200** that causes all slaves in set Ej that do not satisfy product term pi to move to set Ek. Step **2833** issues a command that causes all slaves in set Ej to move to set El. Similarly, step **2832** issues a sequence of commands defined by NAND method **2200** that causes all slaves in set Ek that do not satisfy product term pi to move to set Ej. Step **2834** issues a command that causes all slaves in set Ek to move to set El. The iteration ends after the last product term, pP, has been processed by either steps **2831** and **2833** (odd P case), or steps **2832** and **2834** (even P case). That terminates the method.

If method **2800** is used on the EXCLUSIVE-OR condition ~A*B+A*~B, the sequence of commands **2910** shown in FIG. **29** results. The default initial configuration **2900** where all slaves are in set E**1** is used. In this example, j=1, k=2 and l=3. Rows **2901** and **2902** correspond to the application of method **2200** from E**1** to E**2** to the product term ~A*B. Row **2903** is the accumulation of that product term in E**3**. Rows **2904** and **2905** correspond to the application of method **2200** from E**2** to E**1** to the product term A*~B. Row **2906** is the accumulation of that product term in E**3**. Because the number of product term is even in this case, E**2** is empty at the end.

A similar sequence of commands results if method **2800** is used on the condition A*C+B*C. The sequence of commands **3010** is shown in FIG. **30**. The default initial configuration **3000** where all slaves are in set E**1** is used. In this example, j=1, k=2 and l=3. Rows **3001** and **3002** correspond to the application of method **2200** from E**1** to E**2** to the product term A*C. Row **3003** is the accumulation of that product term in E**3**. Rows **3004** and **3005** correspond to the application of method **2200** from E**2** to E**1** to the product term B*C. Row **3006** is the accumulation of that product term in E**3**. Because the number of product term is even in this case, E**2** is empty at the end.

The method shown in FIG. **31** differs from methods **2500** and **2800** in that both the building of product terms and the accumulation of product terms alternates between two sets. Specifically, the roles of set Ek (state **2**) and El (state **3**) reverse. With method **3100** product terms are either computed from set Ej to set Ek or from set Ej to set El. While for methods **2500** and **2800** summing was done by adding the latest product term into the previous accumulation, with method **3100** the previous accumulation is added to the latest product term. Accordingly, when the latest product term is built in set Ek, accumulation is from El to Ek, and when the latest product term is built in set El, accumulation is from Ek to El. Method **3100** takes as input **3101** the same input as do methods **2500** and **2800**. Method **3100** outputs a configuration whereby all slaves in Ej that satisfy the sum-of-products condition are moved to either set Ek or El depending on the number of product terms. If the condition has an even number of product terms, El will contain the desired slaves; otherwise, Ek will. That can be represented mathematically as

Ej=Ej*~(p**1**+p**2**+ . . . +pP)

(if P is odd) Ek=El+Ej*(p**1**+p**2**+ . . . +pP)

(if P is even) Ek=0

(if P is odd) El=0

(if P is even) El=El+Ej*(p**1**+p**2**+ . . . +pP)

The main step **3120** controls the iteration over all the product terms that make up the sum-of-products condition **3101**. For each product term pi, where i varies from 1 to P, step **3130** tests whether i is odd or even. If i is odd, steps **3131** and **3133** are executed for product term pi. If i is even, steps **3132** and **3134** are executed for product term pi. Step **3131** issues a sequence of commands defined by AND method **2100** that causes all slaves in set Ej that satisfy product term pi to move to set Ek. Step **3133** issues a command that causes all slaves in set El to move to set Ek. Similarly, step **3132** issues a sequence of commands defined by AND method **2100** that causes all slaves in set Ej that satisfy product term pi to move to set El. Step **3134** issues a command that causes all slaves in set Ek to move to set El. The iterative step **3120** ends after the last product term, pP, has been processed by either steps **3131** and **3133** (odd P case), or steps **3132** and **3134** (even P case). That terminates the method.

If method **3100** is used on the EXCLUSIVE-OR condition ~A*B+A*~B, the sequence of commands **3210** shown in FIG. **32** results. The default initial configuration **3200** where all slaves are in set E**1** is used. In this example, j=1, k=2 and l=3. Rows **3201** and **3202** correspond to the application of method **2100** from E**1** to E**2** to the product term ~A*B. Row **3203** is the accumulation of E**3** into that product term. Because E**3** is empty, this command is unnecessary but is included as part of the normal cycle. Rows **3204** and **3205** correspond to the application of method **2100** from E**1** to E**3** to the product term A*~B. Row **3206** is the accumulation of E**2** into that product term. Because the number of product terms is even in this case, E**3** contains the desired set of slaves.

A similar sequence of commands results if method **3100** is used on the condition A*C+B*C. The sequence of commands **3310** is shown in FIG. **33**. The default initial configuration **3300** where all slaves are in set E**1** is used. In this example, j=1, k=2 and l=3. Rows **3301** and **3302** correspond to the application of method **2100** from E**1** to E**2** to the product term A*C. Row **3303** is the accumulation of E**3** into that product term. Because E**3** is empty, this command is unnecessary but is included as part of the normal cycle. Rows **3304** and **3305** correspond to the application of method **2100** from E**1** to E**3** to the product term B*C. Row **3306** is the accumulation of E**2** into that product term. Because the number of product term is even in this case, E**3** contains the desired set of slaves.

The method shown in FIG. **34** differs from methods **2500**, **2800** and **3100** in that the building of product terms and the accumulation of those product terms rotates among three sets, i.e., the states cycle roles. Methods **3400** takes as input **3401** the same input as do methods **2500**, **2800** and **3100**. Method **3400** outputs a configuration whereby all slaves in set Ej that satisfy the sum-of-products condition are moved to set Ej, Ek or El depending on the number of product terms. If the number of product terms modulo **3** is one, that is, 1, 4, 7, etc., set Ej will end up containing the desired set of slaves. If the number of product terms modulo **3** is two, that is, 2, 5, 8, etc., set Ek will end up containing the desired set of slaves. If the number of product terms modulo **3** is zero, that is, 3, 6, 9, etc., set El will end up containing the desired set of slaves. That can be represented mathematically as

(if (P mod **3**)=1) Ej=El+p**1**+p**2**+ . . . +pP, El=0

(if (P mod **3**)=2) Ek=El+p**1**+p**2**+ . . . +pP, Ej=0

(if (P mod **3**)=0) El=El+p**1**+p**2**+ . . . +pP, Ek=0

The main step **3410** controls the iteration over all product terms that make up the sum-of-products condition **3401**. For each product term pi, where i varies from 1 to P, step **3420** tests whether i mod **3** is one, two or zero. If one, steps **3421** and **3424** are executed for product term pi. If two, steps **3422** and **3425** are executed for product term pi. If zero, steps **3423** and **3426** are executed for product term pi. Step **3421** issues a sequence of commands defined by NAND method **2200** that causes all slaves in set Ej that do not satisfy product term pi to move to set Ek. Step **3424** issues a command that causes all slaves in set El to move to set Ej. Similarly, step **3422** issues a sequence of commands defined by NAND method **2200** that causes all slaves in set Ek that do not satisfy product term pi to move to set El. Step **3425** issues a command that causes all slaves in set Ej to move to set Ek. Similarly, step **3423** issues a sequence of commands defined by NAND method **2200** that causes all slaves in set El that do not satisfy product term pi to move to set Ej. Step **3426** issues a command that causes all slaves in set Ek to move to set El. The iterative step **3410** ends after the last product term, pP, has been processed. That terminates the method.

If method **3400** is used on the EXCLUSIVE-OR condition ~A*B+A*~B, the sequence of commands **3510** shown in FIG. **35** results. The default initial configuration **3500** where all slaves are in set E**1** is used. In this example, j=1, k=2 and l=3. Rows **3501** and **3502** correspond to the application of method **2200** from E**1** to E**2** to the negated product term ~(~A*B). Row **3503** is the accumulation of E**3** into E**1**. Because E**3** is empty, this command is unnecessary but is included as part of the normal cycle. Rows **3504** and **3505** correspond to the application of method **2200** from E**2** to E**3** to the negated product term ~(A*~B). Row **3506** is the accumulation of E**1** into E**2**. Because the number of product terms modulo three is two in this case, E**2** contains the desired set of slaves.

A similar sequence of commands results if method **3500** is used on the condition A*C+B*C. The sequence of commands **3610** is shown in FIG. **36**. The default initial configuration **3600** where all slaves are in set E**1** is used. In this example, j=1, k=2 and l=3. Rows **3601** and **3602** correspond to the application of method **2200** from E**1** to E**2** to the negated product term ~(A*C). Row **3603** is the accumulation of E**3** into E**1**. Because E**3** is empty, this command is unnecessary but is included as part of the normal cycle. Rows **3604** and **3605** correspond to the application of method **2200** from E**2** to E**3** to the negated product term ~(B*C). Row **3606** is the accumulation of E**1** into E**2**. Because the number of product terms modulo three is two in this case, E**2** contains the desired set of slaves.

Methods **2500**, **2800**, **3100** and **3400** do not in general generate a minimal sequence of commands for a given arbitrarily complex condition expressed in sum-of-products form. A shorter command sequence can be obtained when an arbitrarily complex condition can be written by an expression that can be generated by the following grammar:

In-expression: (In-expression)*primitive_condition

In-expression: In-expression+primitive_condition

In-expression: primitive_condition

where In-expression is the name given to this kind of expression, namely, left-nesting expression. A In-expression can be written as, ((. . . (((c**1**) op**2** c**2**) op**3** c**3**) . . .)opN cN), wherein c**1**, c**2**, . . . , cN are primitive conditions and op**2**, op**3**, . . . , opN are either *(AND) or +(OR) binary operators. The In-expression as written above is more heavily parenthetically bracketed than necessary and some parenthesis may be deleted as long as the logic is preserved. Left-nesting expressions can be executed using only two of the three states of the three-state machine. An arbitrarily complex condition such as A*B+A*C can be expressed according to the grammar as (B+C)*A. The aforementioned canonical method over the former, sum-of-products, expression requires six commands to execute and uses three states. The latter, left-nesting, expression can be computed with only three commands and uses only two states. Not every arbitrarily complex conditions can be expressed by a single left-nested expression, but any complex condition can be expressed by a sum of left-nested expressions, which requires fewer commands than the canonical sum-of-products form. For example, the condition A*B+A*C+~A*~B+~A*~C can be written as a sum of two left-nested expressions: (B+C)*A+(~B+~C)*~A; the former requires twelve commands, while the latter only eight. As with the canonical sum-of-products method, which uses the third state to accumulate products, the method for executing sum-of-left-nested-expressions uses the third state to accumulate left-nested expressions.

As mentioned above, the present three-selection-state machine is capable of isolating or selecting slaves that satisfy any possible condition expressed by a left-nested expression. Specifically, the invention is necessary and sufficient to isolate and select slaves satisfying those conditions that are expressed by a sum-of-left-nested-expressions. The invention enables this capability because a separate condition (or set of conditions), each corresponding

to a set of slaves, can be isolated in any one of the three states at any given time. Therefore, operations on two sets of conditions, in two of the respective states, can be performed without affecting or being affected by the conditions held in the third state. Specific instances of left-nested-expressions handled by the invention are now presented.

The method for computing the sequence of commands necessary to transfer from a set Ej to a set Ek slaves in set Ej that satisfy a condition given as a In-expression is shown in FIG. **37**. Method **3700** takes as input **3701** two different sets Ej and Ek of the possible three sets E**1**, E**2** and E**3**, and a In-expression, ((. . . (((c**1**) op**2** c**2**) op**3** c**3**) . . .) opN cN). Method **3700** outputs a configuration **3702** whereby all slaves in set Ej that satisfy the In-expression of input **3701** are moved to set Ek. If set Ek was not empty to start with, a side effect of the method is that all slaves originally in set Ek that did not satisfy the reduced condition (. . . ((cM) op . . .) . . . opN cN), where M is such that opM is the first * (AND) operator in the In-expression, have moved to set Ej. That can be represented mathematically as

Ej=Ek*~((cm op . . .) opN cN)+Ej*~((c**1** op . . .) opN cN)

Ek=Ek* ((cm op . . .) opN cN)+Ej* ((c**1** op . . .) opN cN)

where m such that op**2**, op**3**, . . . , opm-1=OR and opm'AND

Method **3700** begins with step **3710**, which issues a command that causes all slaves in set Ej that satisfy the leftmost (first) primitive condition c**1** to move to set Ek. Step **3720** controls the iteration over the binary operators and attendant right operands, from the leftmost to the rightmost, that is, from op**2** to opN and their attendant c**2** to cN. For each operator opi, where i varies from 2 to N, step **3730** tests which binary operator is opi. If opi is the AND operator *, step **3731** is executed; otherwise, opi is the OR operator +, in which case step **3732** is executed. Step **3731** issues a command that causes all slaves in set Ek that do not satisfy primitive condition ci to move to set Ej. Step **3732** issues a command that causes all slaves in set Ej that satisfy primitive condition ci to move to set Ek. After the last iteration, over opN and cN, step **3720** terminates the iteration. That terminates the method.

An important variation of method **3700** is shown in FIG. **38**. Method **3800** computes the sequence of commands necessary to transfer from a set Ej to a set Ek slaves in set Ej that satisfy a condition given as the negation of a In-expression. Method **3800** takes as input **3801** two different sets Ej and Ek of the possible three sets E**1**, E**2** and E**3**, and a condition in the form of a negated In-expression, ~((. . . (((c**1**) op**2** c**2**) op**3** c**3**) . . .) opN cN). Method **3800** outputs a configuration **3802** whereby all slaves in set Ej that satisfy the negated In-expression of input **3801** are moved to set Ek. If set Ek was not empty to start with, a side effect of the method is that all slaves originally in set Ek that did not satisfy the reduced condition ~(. . . ((cM) op . . .) . . . opN cN), where M is such that opM is the first + (OR) operator in the In-expression, have moved to set Ej. That can be represented mathematically as

Ej=Ek* ((cm op . . .) opN cN)+Ej* ((c**1** op . . .) opN cN)

Ek=Ek*~((cm op . . .) opN cN)+Ej*~((c**1** op . . .) opN cN)

where m such that op**2**, op**3**, . . . , opm-1=AND and opm=OR

Method **3800** begins with step **3810**, which issues a command that causes all slaves in set Ej that do not satisfy the leftmost (first) primitive condition c**1** to move to set Ek. Step **3820** controls the iteration over the binary operators and attendant right operands, from the leftmost to the rightmost, that is, from op**2** to opN and their attendant c**2** to cN. For each operator opi, where i varies from 2 to N, step **3830** tests which binary operator is opi. If opi is the OR operator +, step **3831** is executed; otherwise, opi is the AND operator *, in which case step **3832** is executed. Step **3831** issues a command that causes all slaves in set Ek that satisfy primitive condition ci to move to set Ej. Step **3832** issues a command that causes all slaves in set Ej that do not satisfy primitive condition ci to move to set Ek. After the last iteration, over opN and cN, step **3820** terminates. That terminates the method.

By expressing the minimized sum-of-products condition A*C+B*C, used in previous examples, as an In-expression (A+B)*C, either method **3700** or **3800** can be used to generate a sequence of commands that is shorter than the sequence generated by method **2500**, **2800**, **3100** or **3400**. The latter sequence is six commands long, as shown in FIGS. **27**, **30**, **33**, and **36**. Both methods **3700** and **3800** generate a sequence that is three commands long. The example sequence **3910** generated by method **3700** is shown in FIG. **39**. The example sequence **4010** generated by method **3800** is shown in FIG. **40**. They both start with the default initial configuration where all slaves are in set E**1**, as shown in rows **3900** and **4000**. In both examples j=1, k=2 and l=3. Method **3700** generates the sequence T**12**(A), T**12**(B) and T**21**(~C). Commands T**12**(A) and T**12**(B) put the partial condition (A+B) in set E**2** as shown in rows **3901** and **3902**. Command T**21**(~C) results in the desired condition (A+B)*C in set E**2** as shown in row **3903**. Method **3800** generates the sequence T**12**(~A), T**21**(B) and T**12**(~C). Commands T**12**(~A) and T**21**(B) put the partial condition (A+B) in set E**1** as shown in rows **4001** and **4002**. Command T**12**(~C) results in the desired condition (A+B)*C in set E**1** as shown in row **4003**.

Note from the method descriptions of FIGS. **37** and **38**, and the examples of FIGS. **39** and **40**, that the third set E**3** is not used. It is an important property of conditions written as a single In-expression, that they require only two of the three states of a three-state machine. This property permits the use of the third state, that is, of set E**3**, as an accumulator of In-expressions. While not all arbitrarily complex conditions can be expressed by a single In-expression, any arbitrary complex condition can be expressed by a sum-of-In-expressions. It is possible therefore to recode methods **2500**, **2800**, **3100** and **3400** to work on sum-of-In-expressions.

Method **2500** is recoded in FIG. **41** for sum-of-In-expressions. Method **4100** takes as input **4101** three sets Ej, Ek and El, that is, some permutation of sets E**1**, E**2** and E**3**, and a condition written as a sum-of-In-expressions, n**1**+n**2**+ . . . +nN. Assuming, for simplicity, that set Ek is empty at the start of the method. Method **4100** outputs a configuration **4102** whereby all slaves in set Ej that satisfy the sum-of-In-expressions condition **4101** are moved to set El. That can be represented mathematically as

Ej=Ej*~(n**1**+n**2**+ . . . +nN)

Ek=0

El=El+Ej*(n**1**+n**2**+ . . . +nN)

The main step **4120** controls the iteration over all In-expressions of condition **4101**. For each In-expression ni, where i varies from 1 to N, steps **4121** and **4122** are executed in that order. Step **4121** issues a sequence of commands defined by method **3700** that causes all slaves in set Ej that satisfy the In-expression ni to move to set Ek. Step **4122** issues a command that causes all slaves in set Ek to move to

set El. Iteration ends after the last In-expression, nN, has been processed. That terminates the method.

Method **2800** is recoded in FIG. **42** for sum-of-In-expressions. Method **4200** takes as input **4201** the same input as does method **4100**. Method **4200** outputs a configuration **4202** whereby all slaves in Ej that satisfy the sum-of-In-expressions condition **4201** are moved to set El, and either set Ej or set Ek will be empty depending on the number of In-expressions. If the condition has an even number of In-expressions, set Ek will be empty; otherwise, set Ej will be empty. That can be represented mathematically as

El=El+Ej*(n**1**+n**2**+ . . . +nN)

(if N is even) Ej=Ej*~(n**1**+n**2**+ . . . +nN), Ek=0

(if N is odd) Ek=Ej*~(n**1**+n**2**+ . . . +nN), Ej=0

The main step **4220** controls the iteration over all In-expressions of condition **4201**. For each In-expression ni, where i varies from 1 to N, step **4230** tests whether i is odd or even. If i is odd, steps **4231** and **4233** are executed for In-expression ni. If i is even, steps **4232** and **4234** are executed for In-expression ni. Step **4231** issues a sequence of commands defined by method **3800** that causes all slaves in set Ej that do not satisfy In-expression ni to move to set Ek. Step **4233** issues a command that causes all slaves in set Ej to move to set El. Similarly, step **4232** issues a sequence of commands defined by method **3800** that causes all slaves in set Ek that do not satisfy In-expression ni to move to set Ej. Step **4234** issues a command that causes all slaves in set Ek to move to set El. The iteration ends after the last In-expression, nP, has been processed by either steps **4231** and **4233**, or steps **4232** and **4234**. That terminates the method.

Method **3100** is recoded in FIG. **43** for sum-of-In-expressions. Method **4300** takes as input **4301** the same input as do methods **4100** and **4200**. Method **4300** outputs a configuration **4302** whereby all slaves in set Ej that satisfy the sum-of-In-expressions condition **4301** are moved to either set Ek or set El depending on the number of In-expressions. If the condition has an even number of In-expressions, set El will contain the desired slaves; otherwise, set Ek will. That can be represented mathematically as

Ej=Ej*~(n**1**+n**2**+ . . . +nN)

(if N is odd) Ek=El+Ej*(n**1**+n**2**+ . . . +nN), El=0

(if N is even) El=El+Ej*(n**1**+n**2**+ . . . +nN), Ek=0

The main step **4320** controls the iteration over all In-expressions of condition **4301**. For each In-expression ni, where i varies from 1 to N, step **4330** tests whether i is odd or even. If i is odd, steps **4331** and **4333** are executed for In-expression ni. If i is even, steps **4332** and **4334** are executed for In-expression ni. Step **4331** issues a sequence of commands defined by method **3700** that causes all slaves in set Ej that satisfy In-expression ni to move to set Ek. Step **4333** issues a command that causes all slaves in set El to move to set Ek. Similarly, step **4332** issues a sequence of commands defined by method **3700** that causes all slaves in set Ej that satisfy In-expression ni to move to set El. Step **4334** issues a command that causes all slaves in set Ek to move to set El. The iteration ends after the last In-expression, nN, is processed. That terminates the method.

Method **3400** is recoded in FIG. **44** for sum-of-In-expressions. Method **4400** takes as input **4401** the same input as do methods **4100**, **4200** and **4300**. Method **4400** outputs a configuration **4402** whereby all slaves in set Ej that satisfy the sum-of-In-expressions condition **4401** are moved to set Ej, Ek or El depending on the number of In-expressions. If the number of In-expressions modulo three is one, that is, 1, 4, 7, etc., set Ej will end up containing the desired set of slaves. If the number of In-expressions modulo three is two, that is, 2, 5, 8, etc., set Ek will end up containing the desired set of slaves. If the number of In-expressions modulo three is zero, that is, 3, 6, 9, etc., set El will end up containing the desired set of slaves. That can be represented mathematically as

(if (N mod **3**)=1) Ej=El+Ej*(n**1**+n**2**+ . . . +nN), El=0

(if (N mod **3**)=2) Ek=El+Ej*(n**1**+n**2**+ . . . +nN), Ej=0

(if (N mod **3**)=0) El=El+Ej*(n**1**+n**2**+ . . . +nN), Ek=0

The main step **4410** controls the iteration over all In-expressions of condition **4401**. For each In-expression ni, where i varies from 1 to N, steps **4419** and **4420** test whether i mod **3** is one, two or zero. If one, steps **4421** and **4424** are executed for In-expression ni. If two, steps **4422** and **4425** are executed for In-expression ni. If zero, steps **4423** and **4426** are executed for In-expression ni. Step **4421** issues a sequence of commands defined by method **3800** that causes all slaves in set Ej that do not satisfy In-expression ni to move to set Ek. Step **4424** issues a command that causes all slaves in set E**1** to move to set Ej. Similarly, step **4422** issues a sequence of commands defined by method **3800** that causes all slaves in set Ek that do not satisfy In-expression ni to move to set El. Step **4425** issues a command that causes all slaves in set Ej to move to set Ek. Similarly, step **4423** issues a sequence of commands defined by method **3800** that causes all slaves in set El that do not satisfy In-expression ni to move to set Ej. Step **4426** issues a command that causes all slaves in set Ek to move to set El. The iteration ends after the last In-expression, nN, is processed. That terminates the method.

For example, the condition A*B+A*C+~A*~B*~C, which would require ten commands if handled by any of methods **2500**, **2800**, **3100** or **3400**, can be rewritten as (B+C)*A+~A*~B*~C and handled by any of methods **4100**, **4200**, **4300** or **4400**, in which case only eight commands are necessary. In the particular case of method **4100** the eight commands are T**12**(B), T**12**(C), T**21**(~A), T**23**(l), T**12**(~A), T**21**(B), T**21**(C), and T**23**(l).

As evident by the examples and method descriptions, not all possible transitions of the three-state machine need be available. Methods **2500** and **4100** can be executed on any of three-state machines **400**, **500**, **600**, or **700**. Methods **2800** and **4200** can be executed on any of three-state machines **400** or **500**. Methods **3100** and **4300** can be executed on any of three-state machines **400** or **500**. Methods **3400** and **4400** can be executed on any of three-state machines **400**, **500**, **600** and **800**. Therefore for three-state machines **400** and **500**, methods **2500**, **2800**, **3100**, **3400**, **4100**, **4200**, **4300** and **4400** can be used singly or in combination. For three-state machine **600**, methods **2500**, **3400**, **4100** and **4400** can be used singly or in combination. For three-state machine **700**, only methods **2500** and **4100** can be used singly or in combination. For three-state machine **800**, only methods **3400** and **4400** can be used singly or in combination.

Other state machines are possible as long as any of three-state machines **400**, **500**, **600**, **700** or **800** remains a corner-stone of the architecture. More states may be added and different transition combinations can be used, any of

which could be realized by those skilled in the art given the disclosure presented herein, and depending upon the particular specifications desired. Moreover concomitant variations in the methods herein described and the form by which conditions are expressed and input to the methods will immediately become apparent to those skilled in the art. For example, iteration over elements of an In-expression could be handled through recursion instead. They can utilize the teachings of this disclosure to create efficient operative embodiments of the system and methods described and claimed. These embodiments are also within the contemplation of the inventor.

I claim:

1. A state machine slave comprising:

three or more states, being at least a first state, a second state, and a third state, the slave being initially in the first state;

a memory with one or more stored information values;

a receiving unit for receiving one or more commands in a command sequence, each of the commands specifying a "transfer from state", "transfer to state", and a primitive condition; and

a processing unit that causes the slave to move to the second state being the "transfer to state" if the first state is the same as the "transfer from state" and one or more of the information values satisfies the primitive condition, the slave being moved to the third state by another command in the command sequence only if the slave is in the second state, and the slave, once moved into the third state, remaining in the third state.

2. A state machine slave, as in claim **1**, wherein the first and second states reverse roles after an end of one or more subsequences in the sequence of commands.

3. A state machine slave, as in **2**, where the first and second states reverse roles at the end of each subsequence corresponding to a term in a sum of left-nested expressions.

4. A state machine slave, as in claim **3**, where the sum of left-nested expressions is a sum of products.

5. A state machine slave, as in claim **1**, where the second and third states reverse roles after an end of one or more subsequences in the sequence of commands.

6. A state machine slave, as in claim **5**, where the second and third states reverse roles at the end of each subsequence corresponding to a term in a sum of left-nested expressions.

7. A state machine slave, as in claim **6**, where the sum of left-nested expressions is a sum of products.

8. A state machine slave, as in claim **1**, where states cycle roles as follows: the first state assumes the role of second state, the second state assumes the role of the third state, and the third state assumes the role of the first state after an end of one or more subsequences in the sequence of commands.

9. A state machine slave, as in claim **8**, where the first, second, and third states cycle roles at the end of each subsequence corresponding to a term in a sum of left-nested expressions.

10. A state machine slave, as in claim **9**, where the sum of left-nested expressions is a sum of products.

11. A system comprising:

two or more slaves, the slaves comprising:

three or more states, being at least a first state, a second state, and a third state, the slave being initially in the first state;

a memory with one or more stored information values;

a receiving unit for receiving one or more commands in a command sequence, each of the commands specifying a "transfer from state", a "transfer to state", and a primitive condition;

a processing unit that causes the slave to move to the second state being the "transfer to state" if the first state is the same as the "transfer from state" and one or more of the information values satisfies the primitive conditions, the slave being moved to the third state by another command in the command sequence only if the slave is in the second state, and the slave, once moved into the third state, remaining in the third state; and

one or more masters for communicating the commands to all the slaves and determining the sequence of commands in order to select a subset of the slaves that satisfy a condition.

12. A system, as in claim **11**, where the master is a base station, the slaves are radio frequency tags and the base station communicates with the radio frequency tags with a radio frequency signal.

13. A system, as in claim **11**, where the one of the subsequences in the sequence of commands moves one or more first sets of slaves from the first state to the second state, then moves one or more second sets of slaves from the second state to the first state, and then moves the slave remaining in the second state to the third state.

14. A system, as in claim **11**, where the one of the subsequences in the sequence of commands moves one or more first sets of slaves from the first state to the second state, then moves one or more second sets of slaves from the second state to the first state, and then moves the slave remaining in the first state to the third state.

15. A system, as in claim **11**, where the one of the subsequences in the sequence of commands moves one or more first sets of slaves from the first state to the second state and then moves the slaves in the second state to the third state.

16. A system, as in claim **11**, where the one of the subsequences in the sequence of commands moves one or more first set of slaves from the first state to the second state and then moves the slaves in the first state to the third state.

17. A method to generate a sequence of commands that are broadcasted to one or more slaves, the method comprising the steps for:

a. communicating one or more first commands that move zero or more of the slaves from a first state to a second state, the slaves moving from the first state to the second state satisfying a first primitive condition in each respective second command;

b. communicating zero or more second commands that move one or more of the slaves from the second state to the first state, the slaves moving from the first state to the second state satisfying a second primitive condition in each respective second command;

c. repeating steps a and b one or more times;

d. communicating a third command that moves the slaves in the second state into a third state.

18. A state machine slave comprising:

three or more states, being at least a first state, a second state, and a third state, the slave being in the first state;

means for storing one or more stored information values;

means for receiving one or more commands in a command sequence, each command specifying a "transfer from state", a "transfer to state", and a primitive condition; and

means for causing the slave to move to the second state being the "transfer to state" if the first state is the same

as the "transfer from state" and one or more of the information values satisfies the primitive condition, the slave being moved to the third state by another command in the command sequence only if the slave is in the second state, and the slave, once moved into the third state, remaining in the third state.

19. A system comprising:

two or more slaves, the slaves comprising:

three or more states, being at least a first state, a second state, and a third state, the slave being in the first state;

means for storing one or more information values;

means for receiving one or more commands in a command sequence, each command specifying a "transfer from state", a "transfer to state", and primitive condition;

means for causing the slave to move to the second state being the "transfer to state" if the first state is the same as the "transfer from state" and one or more of the information values satisfies the primitive condition, the slave being capable of being moved to the third state by another command in the command sequence only if the slave is in the second state, and the slave, once moved into the third state, remaining in the third state; and

one or more means for communicating each command to all the slaves and determining the sequence of commands in order to select a sub-set of the slaves that satisfy a condition.

* * * * *